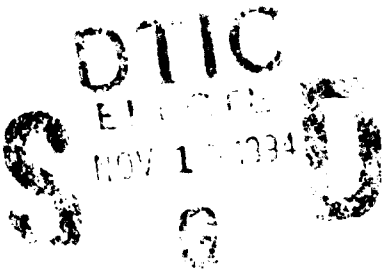
WL-TR-94-1103

EXPERT SYSTEMS IN PREPROCESSING: A PRELIMINARY STUDY OF SUPERVISED LEARNING WITH NOISE USING C4.5

JEFFREY ALAN GOLDMAN



JULY 1994

FINAL REPORT FOR 03/01/94-06/01/94

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

NOTICE

When Government drawings, specifications or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility nor any obligation whatsoever. The fact that the government may have formulated, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise in any manner construed, as licensing the holder or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

JEFFREY ALAN GOLDMAN
Computer Scientist
Technology Section

DOUGLAS S. HAGER, Chief
Technology Section
Target Recognition Technology Branch

WILLIAM E. MOORE, Acting Chief
Mission Avionics Division
Avionics Directorate

If your address has changed, if you wish to be removed from our mailing list or if the addressee is no longer employed by your organization, please notify WL/AART, Bldg 22, 2690 C Street STE 1, Wright-Patterson AFB, OH 45433-7408 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1, 1994	Final March 1994-June 1994

4. TITLE AND SUBTITLE
Expert Systems in Preprocessing: A Preliminary Study of Supervised Learning With Noise Using C4.5

5. FUNDING NUMBERS
PE 61101
PR 0100
TA AA
WU 13

6. AUTHOR(S)
Mr. Jeffrey Alan Goldman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Jeffrey A. Goldman, (513) 255-3215
WL/AART-2 Bldg 22
2690 C St. STE 1
Wright-Patterson AFB OH 45433-7408

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AFB OH 45433-7408

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
WL-TR-94-1103

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The research work in the System Concepts Section is the study of Pattern Theory. The goal of this work is to have a robust method for finding patterns given a set of data samples and from that underlying pattern, to be able to extrapolate the remainder of the function. The work is an intersection of switching theory, computer science, and machine learning. Up until this point, we have not dealt with noisy data. Instead, it was assumed that all of the training data originally given was absolute truth. Furthermore, we have not yet developed any theory about what to do in cases that include noise. Therefore, the goal of this project was to explore some ideas about handling noise quickly in order to gain better insight into the problem. Moreover, this is an area in which I would personally like to pursue a dissertation topic. With the above stated, this project will explore dealing with noise in the domain of binary variables by preprocessing the training data. We will show quantitative results, draw conclusions, and point out future research directions.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Machine Learning, Pattern Theory, Supervised Learning, C4.5 Noise	106
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Contents

List of Tables

1 Introduction

One approach in dealing with noise in any system is to attempt to preprocess the data. This alteration of data is then given to the system with the hopes that we are better off in some way than just having given the system the original data. With this idea, we hoped to do the same for our machine learning methods that are already effective on data void of noise. Specifically, pattern theory is an approach to machine learning that uses function decomposition. However, the software was not designed for multiple samples let alone noise. Also, since we wanted to compare the machine learning method with and without noise, and with and without preprocessing, using our methods would require too much effort for such a limited time. The main problem was that no method currently existed for handling noise to make a baseline comparison. But, the software that is available allows for completely specified testing of several other learning methods originally designed to compare against pattern theory. Thus, it was decided C4.5 [3] would be ideal since it is able to handle noise already. Now we could test C4.5 with noise and with and without preprocessing to determine what methods of preprocessing were most effective.

2 Project Milestones

Before we discuss the steps taken to complete the study, it is necessary to outline the problem framework. The problem is: Given a set of n binary input variables, and one binary output variable, how many samples will we need before we can correctly extrapolate the rest of the data? More generally, how well does the system learn? This will be measured by constructing a learning curve with the axes representing the number of samples given for training and the number of errors on the entire function.

Up until now, these experiments were conducted on various sample sizes without replacement and without noise. The completion of this project now encorporates both abilities. When we refer to noise in this system, we mean for a given input n-tuple, a noisy sample will have an output value flipped, i.e., a 0 becomes a 1 and vice-versa.

The first step in this project was to think of methods of preprocessing the data. One obvious approach was to eliminate any conflict in the training set. In other words, if there were two or more occurrences of the same input n-tuple but their outputs were not all the same value, then throw out all of these data elements from the training set. For example, if we only had one variable and our data set had the elements 0=input, 1=output and 0=input, 0=output, we would throw out both elements. We will refer to this method as "lose-conflicts."

Another method that seemed more promising was to tally up all such duplicates of input data and choose the majority occurring pair. In other words, if for a given input we had 10 outputs, 8 being 1 and only 2 being 0, we would choose the 8 with a 1 and throw out the others. The appeal to this method was that for a very large number of samples, probabilistically speaking, we would ultimately always choose the correct input/output pair. We will refer to this method as "majority-rules."

One final method was a threshold. The method was intended to choose a data element only if it was "strong." In other words, if our original database had missing fields, for a given input value, we may not be able to determine an output value. The threshold would only accept data points if they occurred more than X% of the time of the given training set. For binary variables, this seemed irrelevant. However, if we were considering discrete valued variables, the threshold method would have more meaning.

At this point the expert system was constructed to preprocess data. The program listing and comments are included in Appendix A. The system would be given a set of data points (input/output pairs) and would return an altered data set. The above methods were very simple to encode in CLIPS, an expert system shell developed by NASA. It was used as a rapid prototyping tool. The next step was to implement the necessary routines into our software in order to generate learning curve experiment data.

2.1 FLASH

FLASH is an acronym for Function Learning and Synthesis Hotbed. It can be used for many things, among them the ability to generate learning curves for various learning methods including C4.5 and pattern theory.

	Original Function
F_1	$= (x_1 x_3) + \overline{x}_2$
F_2	$= (x_1 \overline{x}_2 x_3)(x_4 + \overline{x}_6)$
F_3	$= (\overline{x_1 + x_2}) + (\overline{x}_1 x_4 x_6)$
F_4	$= \overline{x}_4$
F_5	$= (x_1 x_2 \overline{x}_4) + (x_3 \overline{x}_5 x_7 x_8) + (x_1 x_2 x_5 x_6 x_8) + (\overline{x}_3 \overline{x}_5)$
F_6	$= x_2 + x_4 + x_6 + x_8$
F_7	$= (x_1 x_2) + (x_3 x_4) + (x_5 x_6) + (x_7 x_8)$
F_8	$= (x_1 \overline{x}_2)$ XOR $(x_1 x_5)$
F_9	$= (x_2$ XOR $x_4)(\overline{x}_1$ XOR $(x_5 x_7 x_8))$
F_{10}	$= (x_1 \Rightarrow x_4)$ XOR $(\overline{x_7 x_8}(x_2 + x_3))$

Table 1: The Functions Tested

FLASH, however, was lacking in the abilities we needed to test noise. Thus, the following routines were added to FLASH as part of this project.

FLASH can now sample with replacement. Instead of giving a set of unique inputs for training, it is now possible to ask for a set of k samples that may include duplicates.

FLASH can now introduce X% noise into the training set. Specifically, after n samples are chosen for training, FLASH generates a random number for each of the n samples. If that number is less than or equal to X, then the output bit is flipped.

FLASH can run a learning experiment given any sample size. This advantage may appear awkward but a necessity arose that when sampling with replacement and with noise, one would need a very large sample size. For example, with an 8 variable function, there are only 2^8=256 possible input samples. However, sampling with replacement and noise, it would be necessary to have as many as 1250 samples to train with. In reality, people who are doing these noise type experiments would actually have a number of samples as large as 5 times the sample size or more.

FLASH can preprocess noisy training data as in the "lose-conflicts" or "majority-rules" methods. It can choose one of these or none at all.

3 Experimental Design

At this point, we are at the stage where we can generate learning curves as we have in the past with FLASH; however, we have new features that allow us to obtain some experimental results dealing with noise and preprocessing. A discussion of the experimental methodology follows.

We wanted to test to see if preprocessing of any kind improves learning performance. We picked a set of 10 functions on 8 binary variables defined in Table 1. These 10 functions were created originally as an attempt to model relationships that might occur in a database [2]. KDD stood for knowledge discovery in databases. C4.5 was then given a set of samples to train on and it was compared to the entire function. The training samples it was given was random. For each point on the graph, (each sample size) C4.5 was tested 10 separate times yielding a maximum, minimum, and average number of errors. One option C4.5 was given was -t 10 meaning 10 trees were iteratively built and the best tree was selected based on the internal testing of the training data. It was also given the option -m 0 which meant the training data had little if any noise.

There are two sets of 10 graphs in Appendix B. The first set tests the 10 functions with sample size ranging from 50 to 1250 samples. The options given were sampling with replacement, 20% noise, and no pruning. Three curves were plotted together for comparison. They are no preprocessing, "lose-conflicts," and "majority-rules."

The second set of ten graphs also tests the 10 functions. The sample size ranges from 50 to 1250 samples. Also, the options of sampling with replacement and 20% noise were given. The only difference was this time, we would compare C4.5's internal noise handling with the other two preprocessing methods. Thus, the line where there was no preprocessing had pruning turned on. C4.5 uses pruning in order to handle noise and to prevent overfitting the data. Pruning is not used if the data given is correct. Since preprocessing is intended to give correct data as best it can, it made sense to have pruning off for the two preprocessing methods when comparing. In other words, it would be strange to preprocess the data and then prune the tree. Moreover, C4.5 gives better performance on correct data without pruning than with pruning.

Both sets plot the average number of errors over 10 trials for a given sample size.

4 Quantitative Results

The hypothesis about preprocessing was that ranging from the worst to the best, the order of performance would be no preprocessing, "lose-conflicts," and "majority-rules." However, the experiments showed the exact opposite!

Comparing the individual functions with the varying preprocessing methods, the best performance (best learning curve) characterised by reducing the number of errors with less samples, was to not have any preprocessing at all. In other words, C4.5's internal ability to handle noise was better than any preprocessing we had done. "Lose-conflicts" was second and the method we thought would be the best, "majority rules," was dead last.

Some reasons we have for the unexpected outcomes are only speculative. The first such speculation is that we are making an implicit assumption that one can actually look at n samples of input x and conclude the majority of the samples must be correct, thus discarding the other data. This would be provably true if we have a very large number of samples by the way we are introducing noise. However, if we do not have enough samples, by preprocessing in any method, we are losing information.

The "lose-conflicts" method discards several data points in the hope that this would be better than keeping the noise in the system. The problem is that C4.5 is able to handle the noise and removing the data is to its disadvantage. The "majority-rules" method performs worse than both because we are speculating that one incorrect training data element is worse than 10 unknown data elements. In other words, if there happens to be a case where the "majority-rules" picks a data point that is in the majority but is the wrong output, C4.5 has little hope of recovery. Moreover, C4.5 is able to learn the function with the noise before we get to a sample size where the majority is provably correct.

Because "majority-rules" has the potential to introduce false information, we feel this is the reason it performed the poorest. "Lose-conflicts," although, no false information is introduced, too much information is lost. Finally, the best method is to hand all of the information to C4.5 in the first place.

5 Conclusions

An expert system was developed in order to test different rules for preprocessing data. Once it was built, those methods were encorporated into our hot bed for testing learning methods. At that point, it was possible to produce learning curves for several parameters.

In conclusion, our experiments demonstrated C4.5's learning ability with noise. Out of the three preprocessing methods we tested "majority-rules," "lose-conflicts," and no preprocessing were given in order of increasing performance. We speculate that the reasons for this are owed to C4.5's powerful internal ability to handle noise without any help needed from preprocessing.

6 Future Work

As a result from all of this, future work would include making the few small steps required so we can test pattern theory with preprocessing. We will not have a ruler for which to measure its performance as with C4.5.

However, it will be possible to compare pattern theory with some preprocessing ("lose-conflicts" or "majority-rules") and C4.5 stand alone. In other words, we can compare the two learning methods not only over several different functions but also with or without noise. We would expect C4.5 to perform better because pattern theory currently has no internal methods for dealing with noise. The best it can do most likely is to throw out any conflicts ("lose-conflicts") and then train on the remaining, speculatively true data.

Although the above is interesting, the real future work that is needed is to develop internal methods of pattern theory (function decomposition) to handle noise in the data as well as in the inputs. One stepping stone required is first to be able to handle missing values in the inputs. Being able to accomplish both, would yield a learning method that is truly robust.

7 Overall Lessons Learned

This effort provided several useful points for later work. The first is defining the whole noise problem framework. Before any work was done, it had to be decided what it means to have noise in the domain. For example, it was clear that if we had sampling without replacement, there is no hope for recovery if any of the samples became noisy. It was also important to define noise in a binary output to mean flipping the bit value. With signal, noise usually refers to information added to it. Some thought was given to noise in the inputs, however, this complicated the problem too much for our time constraints. The model of creating noise among the training set was also thought out. For example, we may have decided that noise could be added to the entire set (test set). In other words, when samples were drawn for training, the errors would be consistent every time. However, there is no hope of recovery in this model unless it is implied that multiple entries exist, some with noise, and others without. With that caveat, however, we are talking about the same problem as our chosen model of sampling with replacement and adding noise to the training set. This exercise, if nothing else, provided a forum for which to explore noise in more detail.

Another useful point to come out of this endeavor is that since we have seen how much better internal noise handling is than preprocessing with C4.5, we can only expect a limited amount of success for such preprocessing in pattern theory. Instead, it would be better to explore internal noise handling. This may be accomplished in several ways but the most obvious choice is within the partition selection of function decomposition.

Converting expert system rules into a standard programming language is a non-trivial task. This is an important lesson to be learned. For example, in writing the rule to "majority-rules," it was very simple in the expert system to ask it to look at all pairs of inputs, deleting one or more if necessary. However, in the already existing code for FLASH, it was necessary to run through a list in some strange way paying very special attention to implementation details. This is not to say that the expert system was not useful. On the contrary, it made it very easy to try new things experimentally. An equally important lesson is that the expert system is a valuable prototyping tool.

Other important lessons and useful results from this project was to get a good "hands-on" feel for the FLASH software. By working with the code directly, I better understood some of the interworkings of this gargantuan 20,000-line program. My updates to the code will be included as part of the continuing evolvement of FLASH.

One final important result to come out of this project is that we are one small step away from testing noise with pattern theory. Already, we can test C4.5 with replacement and noise. In order to be thorough, these changes to the FLASH code were necessary. Moreover, I have a better understanding for a dissertation area of study within this domain.

References

[1] Jeffrey A. Goldman. Machine learning: A comparative study of pattern theory and C4.5. Technical Report WL-TR-94-1102, Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1994. work in progress.

[2] Jeffrey A. Goldman. Pattern theoretic knowledge discovery. In *6th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, November 1994.

[3] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, Palo Alto, California, 1993.

[4] Timothy D. Ross, Michael J. Noviskey, Timothy N. Taylor, and David A. Gadd. Pattern theory: An engineering paradigm for algorithm design. Final Technical Report WL-TR-91-1060, Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1991.

8 Appendices

To be thorough, it was decided to put some of the additional graphs that were made as part of the overall effort.

Here we mention some other works that are relevant to this report. In Goldman, there are extensive tests comparing C4.5 to Pattern Theory [1]. To read more about Pattern Theory specifically, the reader is referred to [4].

A Expert System Code written for CLIPS

Appendix A is the expert system code written in CLIPS, a NASA expert system shell. This code was the used for the rapid testing of preprocessing ideas.

Jeff Goldman
A Data preprocessor in CLIPS

6/1/94

Deftemplates

The Data template has a name of the field (not necessarily unique), a unique id number relative to a given Name, an output value, the number of occurances, and the number of differences.

For example, a data element can be:

```
Name:    Weight-Loss
Id:      1
Output: 10
Occurances: 1
Differences: 0
```

and

```
Name:    Weight-Loss
Id:      2
Output: 20
Occurances: 1
Differences: 0
```

The occurances and differences fields will be calculated by the system. It is assumed that the differences field is always entered initially as 0.

With our research, an output will be a 0 or a 1 and the name will be generic. The Id's are provided for clips so that we may distinguish between facts

```
deftemplate Data
       (slot Name)
       (slot Id)
       (slot Output)
       (slot Occurances)
       (slot Differences))
```


The Noise-Marked template will be used in the Noise-Tally rule. We do not want to add up "differences" forever. Noise-Marked records the fact that id First and id Second were already compared.

```
deftemplate Noise-Marked
```

```
        (slot First)
        (slot Second))

;
;
; DefRules
;
;


; Tally-Equal-Data looks for pairs of data elements that are different
; i.e have unique id's, have the same name, and have the same output.
;
; Once such a pair is found, we combine the occurances and differences.
; Then we delete the other.  In other words, this rule takes and
; tallies all equal sets of data points.
;

(defrule Tally-Equal-Data
        (declare (salienc~ 50))
        (Data (Name ?poi..-A)
              (Id ?id-A)
              (Output ?out-A)
              (Occurances ?occ-A)
              (Differences ?diff-A))
        (Data (Name ?point-A)
              (Id ?id-B&~?id-A)
              (Output ?out-A)
              (Occurances ?occ-B)
              (Differences ?diff-B))
        ?f1 <- (Data (Name ?point-A)
                     (Id ?id-A)
                     (Output ?out-A)
                     (Occurances ?occ-A)
                     (Differences ?diff-A))
        ?f2 <- (Data (Name ?point-A)
                     (Id ?id-B)
                     (Output ?out-A)
                     (Occurances ?occ-B)
                     (Differences ?diff-B))
        =>
        (modify ?f1 (Occurances (+ ?occ-A ?occ-B)))
        (retract ?f2))


;
; Tally-Noise is the equivalent of Tally-Data except here we are looking
; for data points that have the same name but different outputs (noise)
;
; We do not delete any data here.  Instead, we record when there is
; such a pair in a given elements "differences" field.
;
; To insure that we do not tally pairs more than once, we record
; whether or not a pair has been tallied with "Noise-Marked"
;
;


(defrule Tally-Noise                    8
        (declare (salience 40))
        (Data (Name ?point-A)
              (Id ?id-A)
              (Output ?out-A)
              (Occurances ?occ-A)
              (Differences ?diff-A))
        (Data (Name ?point-A)
```

```
            (Id ?id-B&~id-A)
            (Output ?out-B&~?out-A)
            (Occurances ?occ-B)
            (Differences ?diff-B))
        (not (Noise-Marked (First ?id-A)
                           (Second ?id-B)))
        ?f1 <- (Data (Name ?point-A)
                     (Id ?id-A)
                     (Output ?out-A)
                     (Occurances ?occ-A)
                     (Differences ?diff-A))
        ?f2 <- (Data (Name ?point-A)
                     (Id ?id-B)
                     (Output ?out-B)
                     (Occurances ?occ-B)
                     (Differences ?diff-B))
        =>
        (assert (Noise-Marked (First ?id-A) (Second ?id-B)))
        (modify ?f1 (Differences (+ ?diff-A ?occ-B))))


;
; Print-Data prints out every point when all of the modifications
; are complete.
;

(defrule Print-Data
        (Data (Name ?point-A)
              (Id ?id-A)
              (Output ?out-A)
              (Occurances ?occ-A)
              (Differences ?diff-A))
        =>
        (printout t "Name:" ?point-A " id:" ?id-A " Out:" ?out-A " occ:"
                ?occ-A " diff:" ?diff-A crlf))

;
;
; Methods of Preprocessing
;
;


;
; Majority-Rules is a method of data preprocessing that looks at the
; number of occurances for any pair of points that have the same
; inputs but different outputs, and selects the one with more occurances.
;
; In the case when there is a tie, it picks the first one in the pattern
; matching.  This ends up being the first one in the order of the data
; set as given in the initial facts (provided all clips setttings are
; set toto default)
;

(defrule Majority-Rules
        (declare (salience 10))
        (Majority True)
        (Data (Name ?point-A)
              (Id ?id-A)
              (Output ?out-A)
              (Occurances ?occ-A)
              (Differences ?diff-A))
        (Data (Name ?point-A)
              (Id ?id-B&~id-A)
              (Output ?out-B&~?out-A)
              (Occurances ?occ-B)
              (Differences ?diff-B))
```

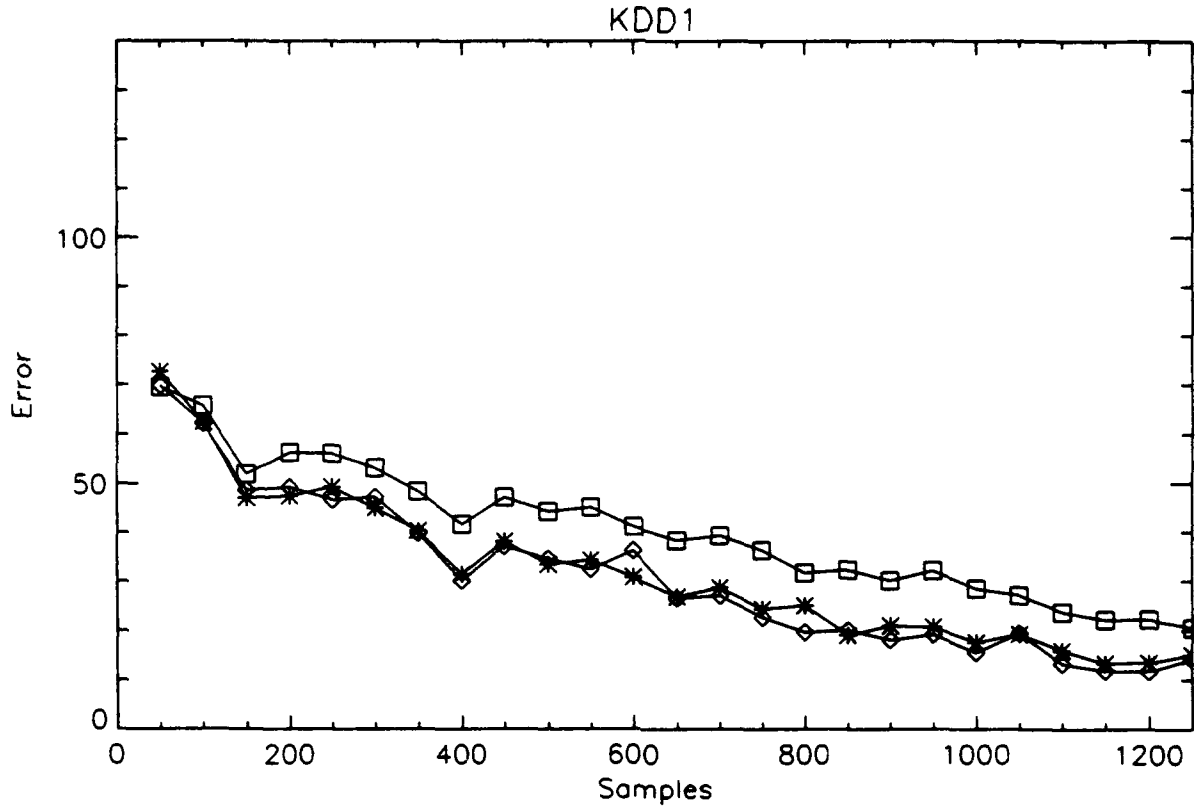
```
        ?f2 <- (Data (Name ?point-A)
                     (Id ?id-B)
                     (Output ?out-B)
                     (Occurances ?occ-B)
                     (Differences ?diff-B))
        (test (>= ?occ-A ?occ-B))
        =>
        (retract ?f2))

;
; Threshold-Rule eliminates any data such that:
;
;       #of occurances
;       --------------                    <       Threshold Value
; (#of occurances + #of differences)
;
;
; The Threshold value is a number between 0-1 provided by the user
;

(defrule Threshold-Rule
        (declare (salience 10))
        (Threshold True)
        (Threshold-Value ?val)
        (Data (Name ?point)
              (Id ?id)
              (Output ?out)
              (Occurances ?occ)
              (Differences ?diff))
        (test (< (/ ?occ (+ ?occ ?diff)) ?val))
        ?f1 <- (Data (Name ?point)
                     (Id ?id)
                     (Output ?out)
                     (Occurances ?occ)
                     (Differences ?diff))
        =>
        (retract ?f1))

;
; Majority+Threshold-Rule is a combination of Majority Rules and
; Threshold.  The data point is required to occur the most of any other
; such point and it's precentage must be >= a given threshold.
;
;
; The purpose of this method is to choose the point that occurs the most
; often within the noise but to only use it if it occurs more than
; some percentage of the time.
;

(defrule Majority+Threshold-Rule
        (declare (salience 10))
        (Majority+Threshold True)
        (Threshold-Value ?val)
        =>
        (assert (Threshold True))
        (assert (Majority True)))

;
; In the No-Noise-Rule, any data point that has a conflict is eliminated.
;
; This amounts to checking the Difference field to see if it is not 0.
;

(defrule No-Noise-Rule
        (declare (salience 10))
        (No-Noise True)
```

```
        (Data (Name ?point)
              (Id ?id)
              (Output ?out)
              (Occurances ?occ)
              (Differences ?diff&~0))
        ?f1 <- (Data (Name ?point)
                     (Id ?id)
                     (Output ?out)
                     (Occurances ?occ)
                     (Differences ?diff))
        =>
        (retract ?f1))

;
;
; Deffacts
;
;
;



;
; Here, we declare which type of preprocessing we wish to use.
;

;(deffacts Noise-Method
;       (No-Noise True))
;       (Majority+Threshold True)
;       (Threshold-Value 0.5))
;       (Majority True))

;
;
; The following is some sample data to ensure that the method is
; working properly
;
;

(deffacts Sample-Data
        (Data   (Name A)
                (Id 1)
                (Output 0)
                (Occurances 1)
                (Differences 0))
        (Data   (Name A)
                (Id 2)
                (Output 0)
                (Occurances 1)
                (Differences 0))
        (Data   (Name A)
                (Id 3)
                (Output 0)
                (Occurances 1)
                (Differences 0))
        (Data   (Name A)
                (Id 4)
                (Output 0)
                (Occurances 1)
                (Differences 0))
        (Data   (Name A)
                (Id 5)
                (Output 1)
                (Occurances 1)
                (Differences 0))
        (Data   (Name A)
                (Id 6)
```

```
          (Output 1)
          (Occurances 1)
          (Differences 0))
(Data     (Name A)
          (Id 7)
          (Output 2)
          (Occurances 1)
          (Differences 0))
(Data     (Name A)
          (Id 8)
          (Output 2)
          (Occurances 1)
          (Differences 0))
(Data     (Name A)
          (Id 9)
          (Output 2)
          (Occurances 1)
          (Differences 0))
(Data     (Name B)
          (Id 1)
          (Output 0)
          (Occurances 1)
          (Differences 0))
(Data     (Name B)
          (Id 2)
          (Output 1)
          (Occurances 1)
          (Differences 0))
(Data     (Name C)
          (Id 1)
          (Output 0)
          (Occurances 1)
          (Differences 0))
(Data     (Name D)
          (Id 1)
          (Output 0)
          (Occurances 1)
          (Differences 0)))
```

B C4.5 Comparison Graphs of Different Preprocessing Methods

Appendix B shows the graphs discussed in the body of the report. These are what the main results are based on.



Comparison of preprocessing methods

No Preprocessing (average error)

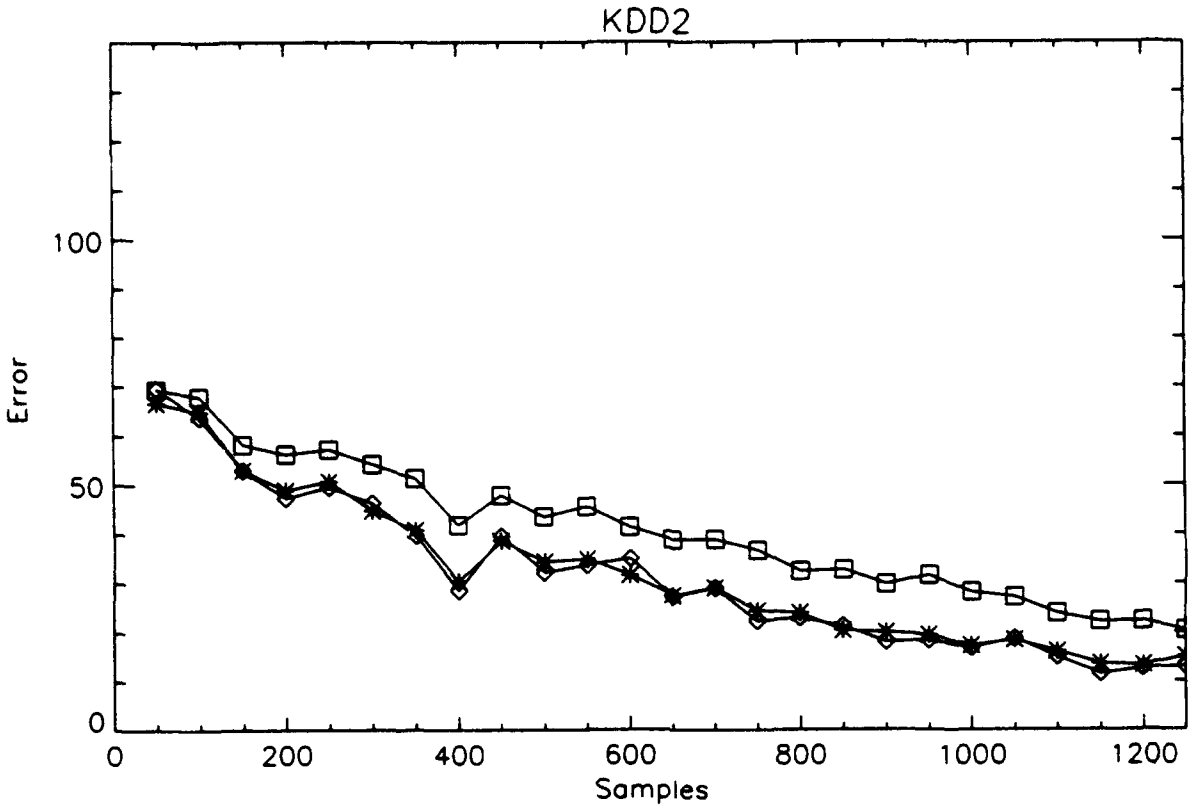
Lose Conflicts (average error)

Majority Rules (average error)

C45 −t10 −m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

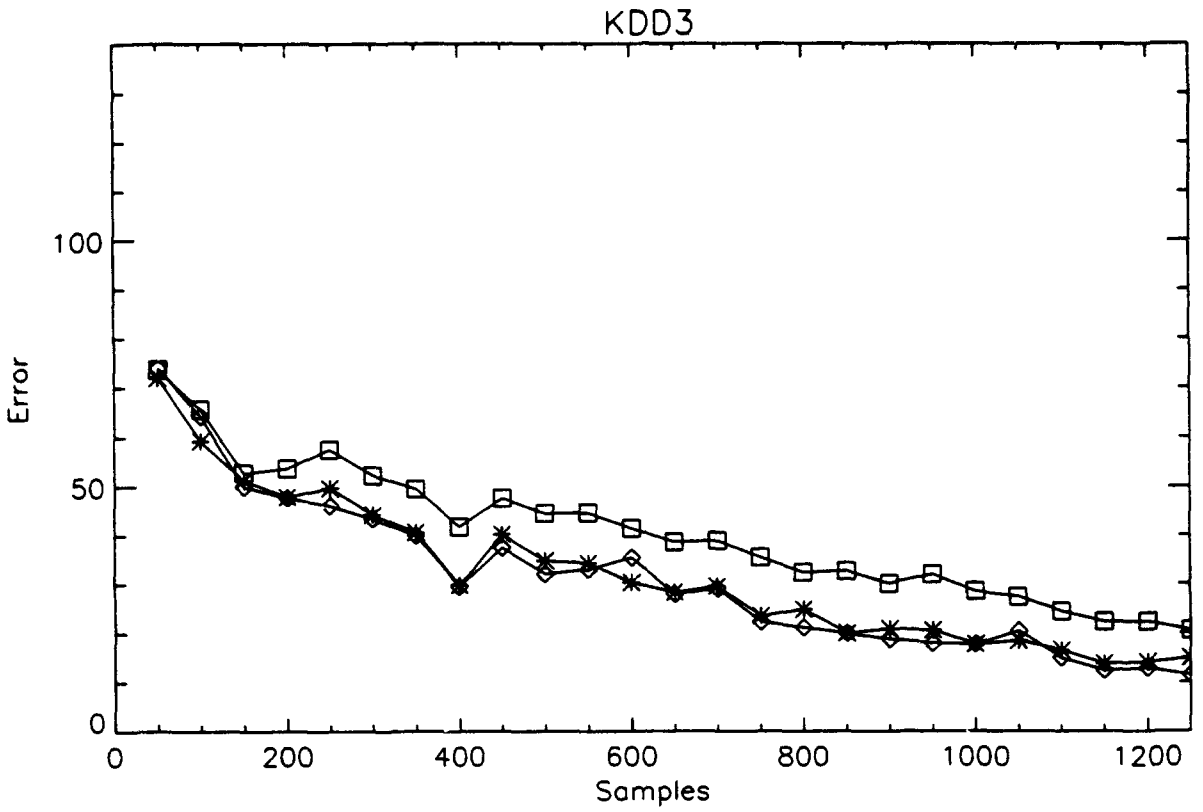
Lose Conflicts (average error)

Majority Rules (average error)

C45 −t10 −m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

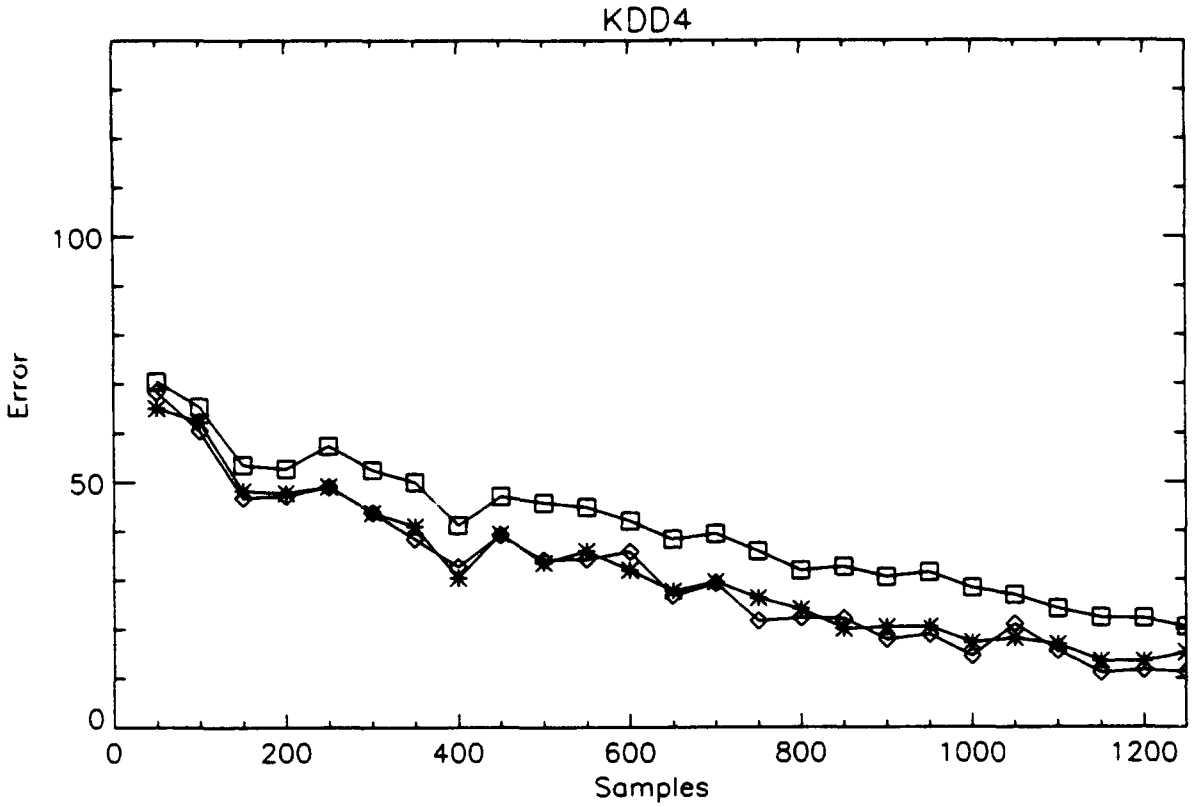
Lose Conflicts (average error)

Majority Rules (average error)

C45 -t10 -m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

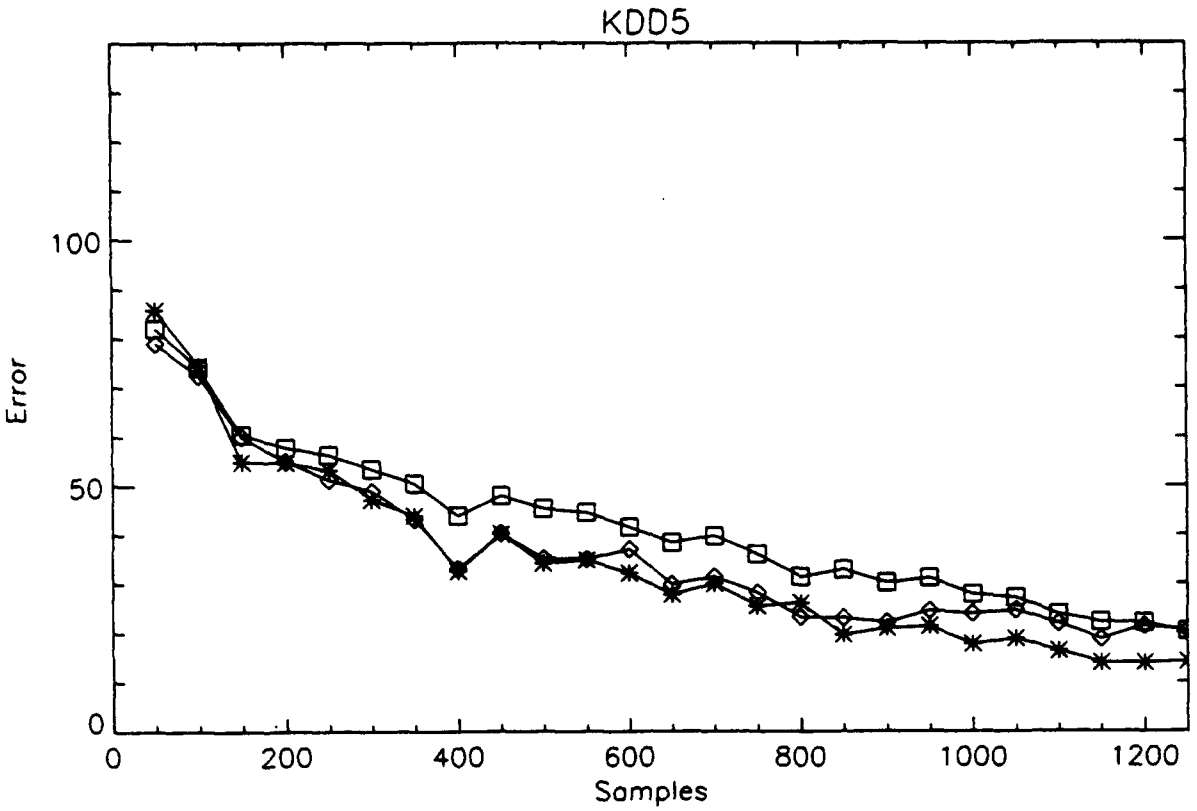
Lose Conflicts (average error)

Majority Rules (average error)

C45 −t10 −m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

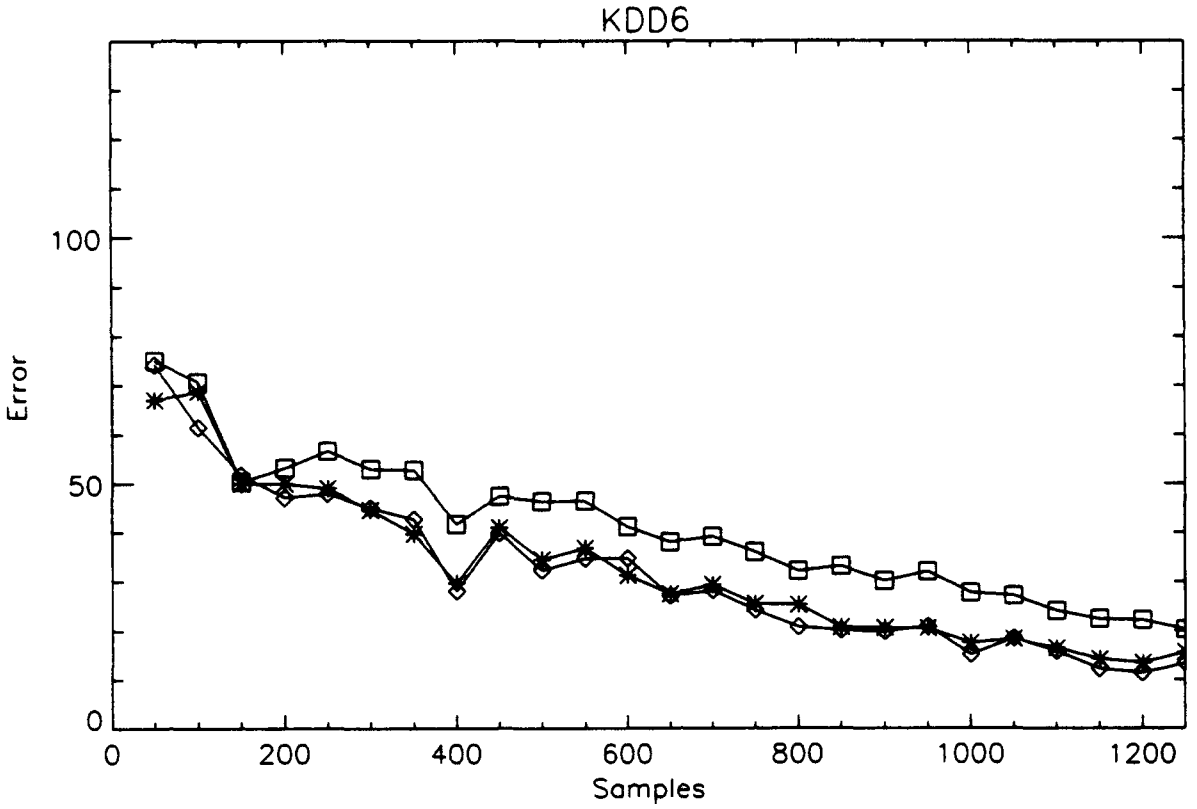
Lose Conflicts (average error)

Majority Rules (average error)

C45 −t10 −m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

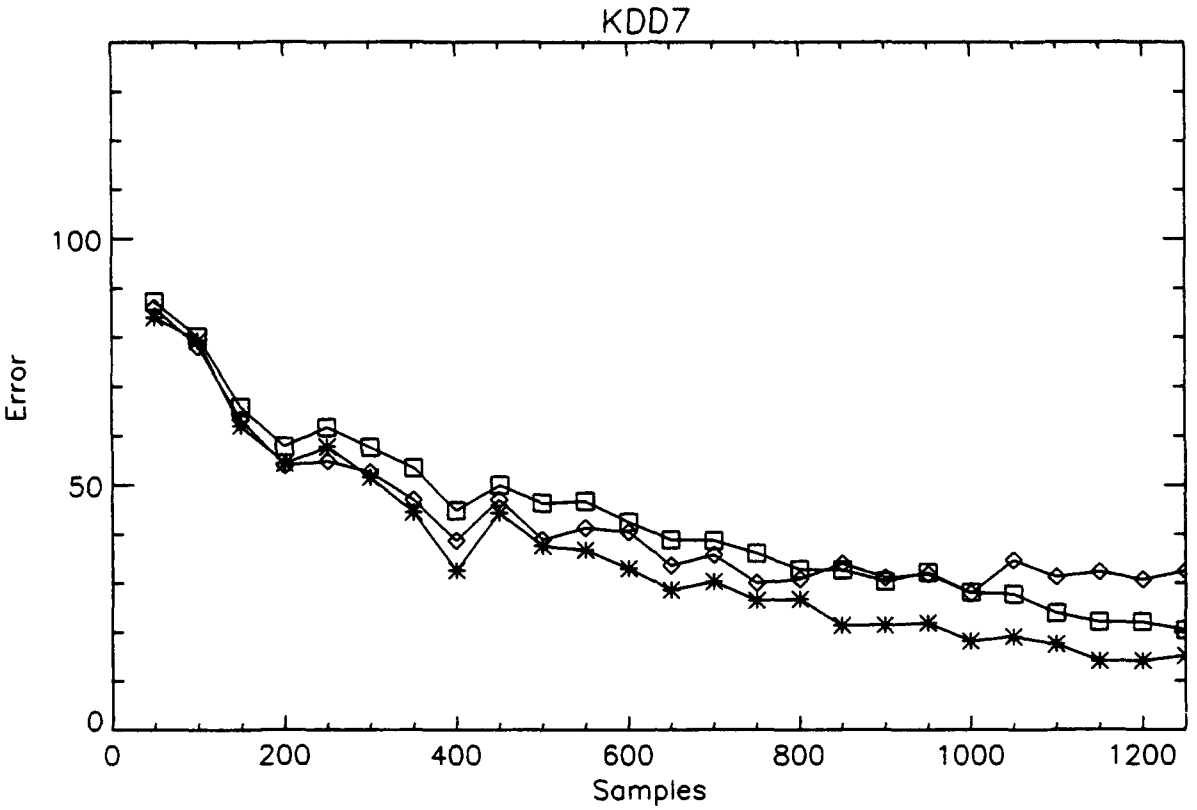
Lose Conflicts (average error)

Majority Rules (average error)

C45 −t10 −m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

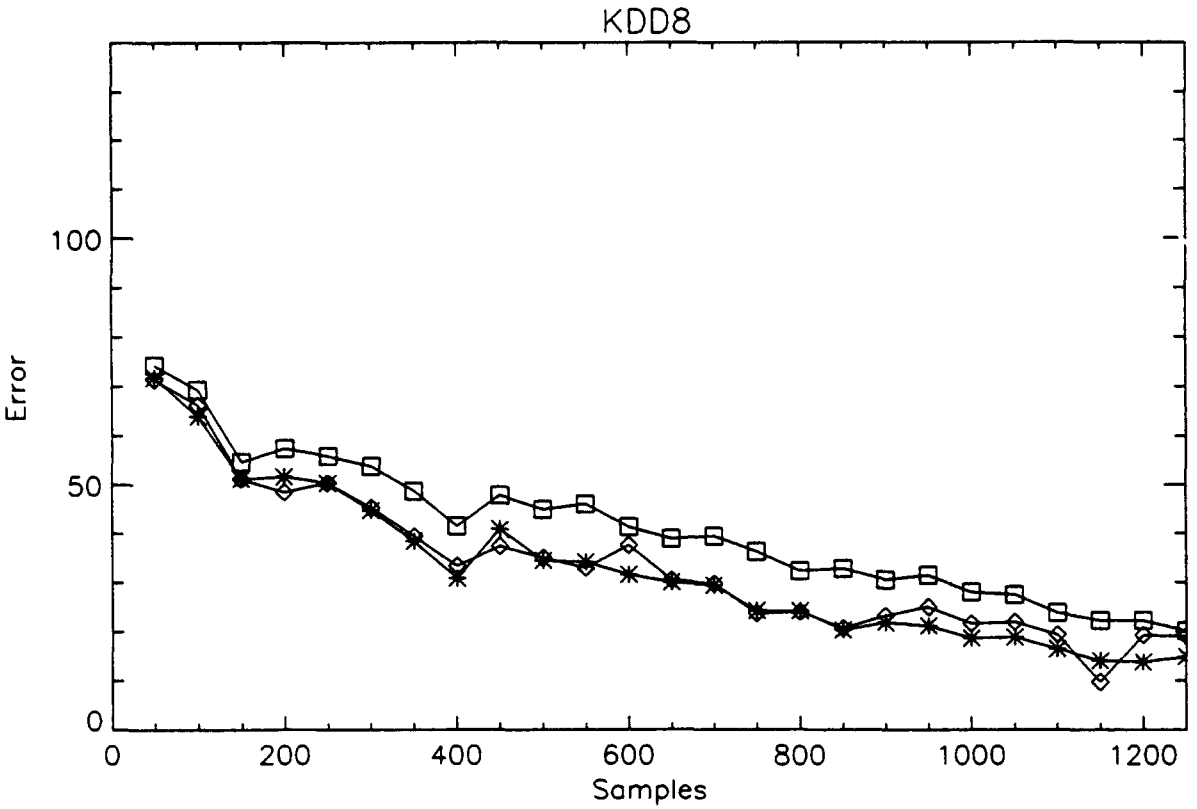
Lose Conflicts (average error)

Majority Rules (average error)

C45 −t10 −m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

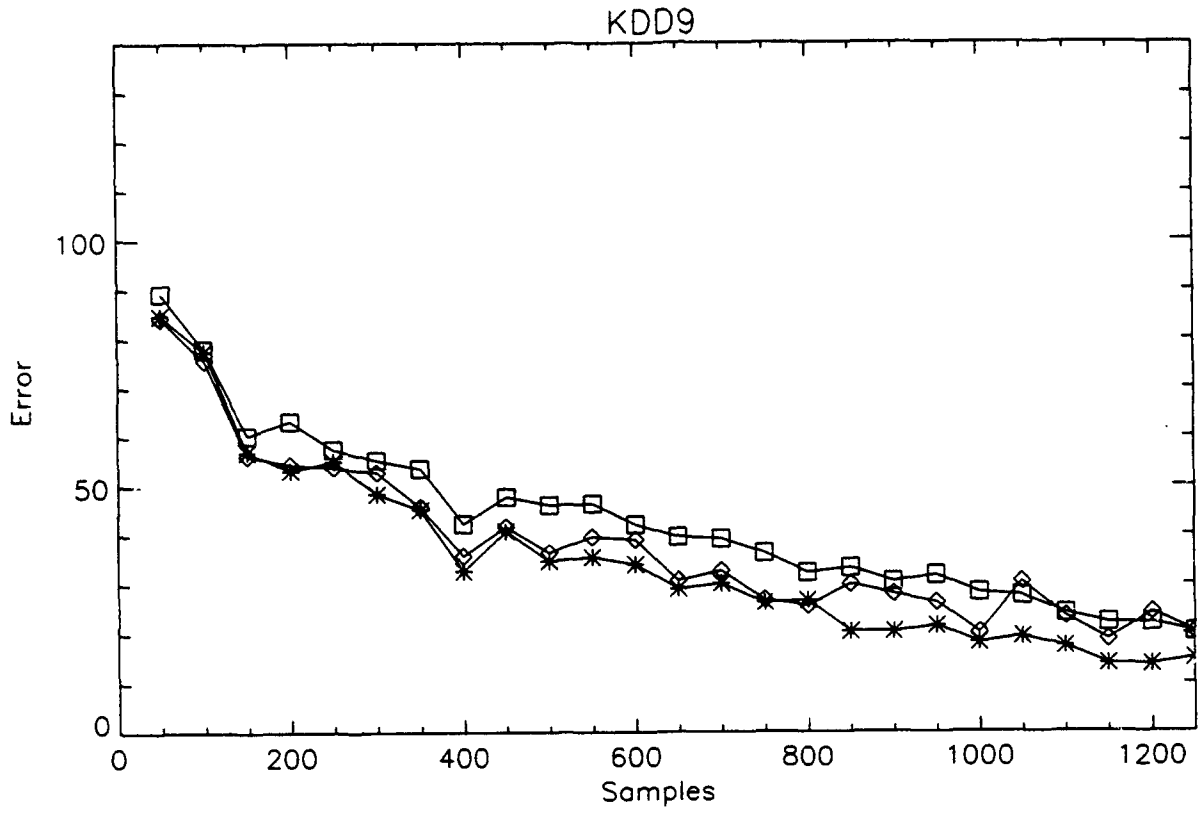
Lose Conflicts (average error)

Majority Rules (average error)

C45 −t10 −m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

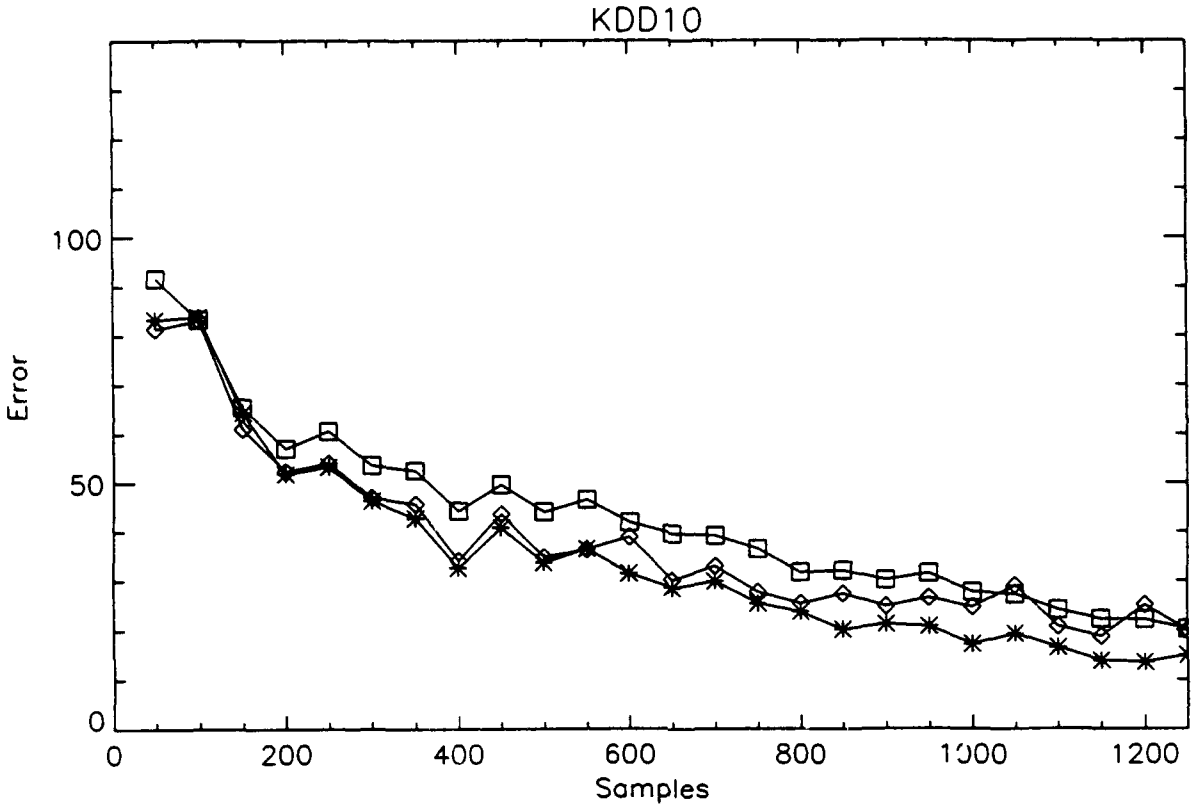
Lose Conflicts (average error)

Majority Rules (average error)

C45 -t10 -m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing (average error)

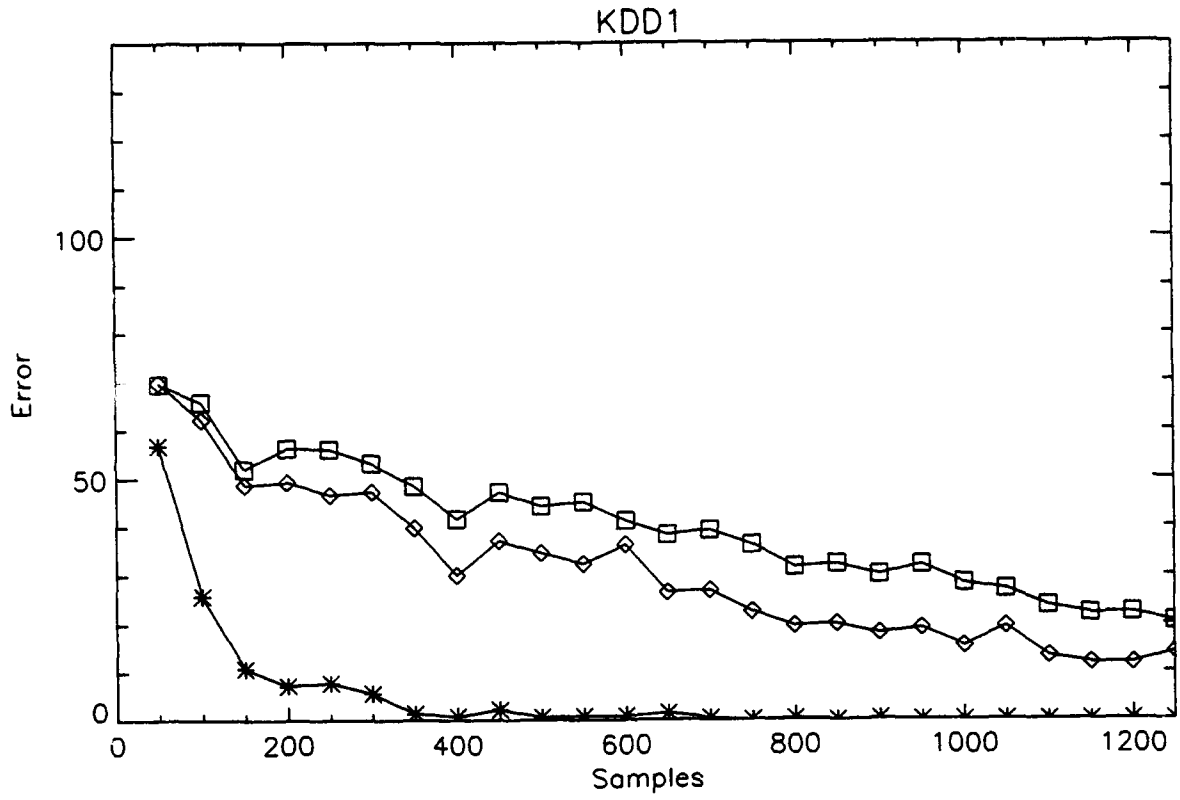
Lose Conflicts (average error)

Majority Rules (average error)

C45 −t10 −m0 no pruning

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

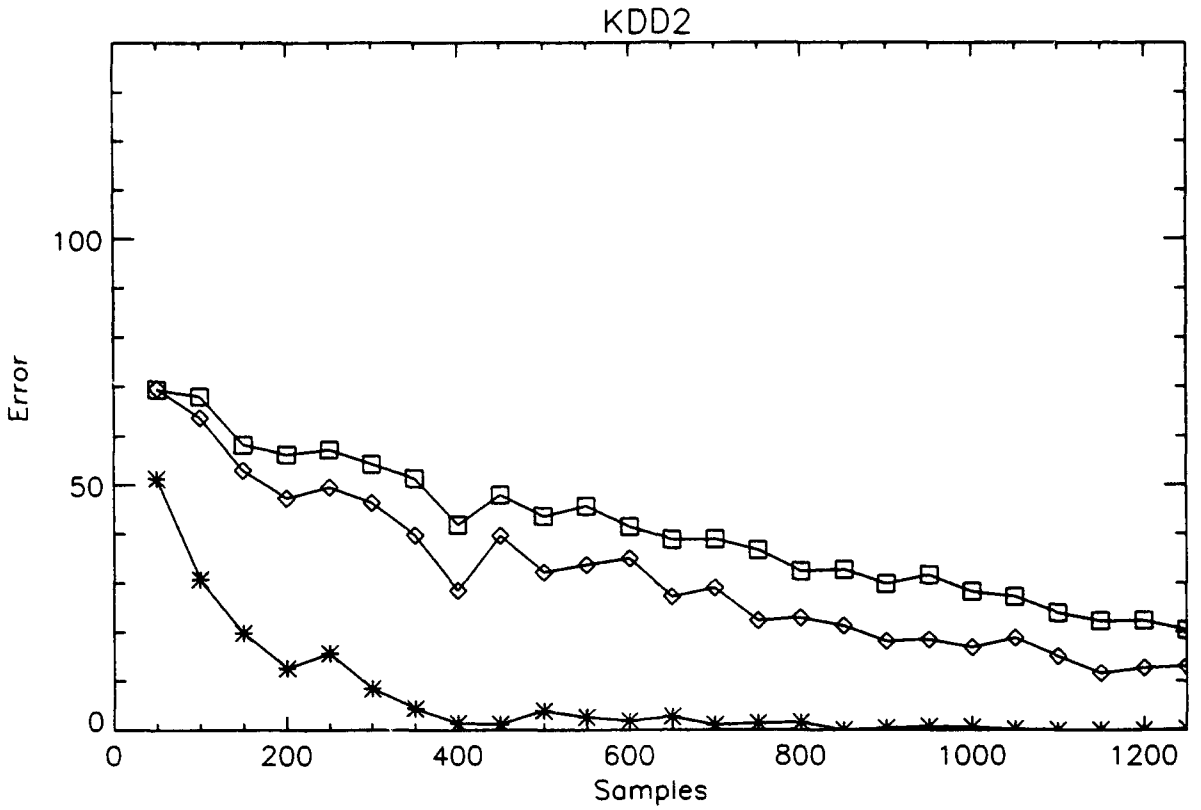
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 −t10 −m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

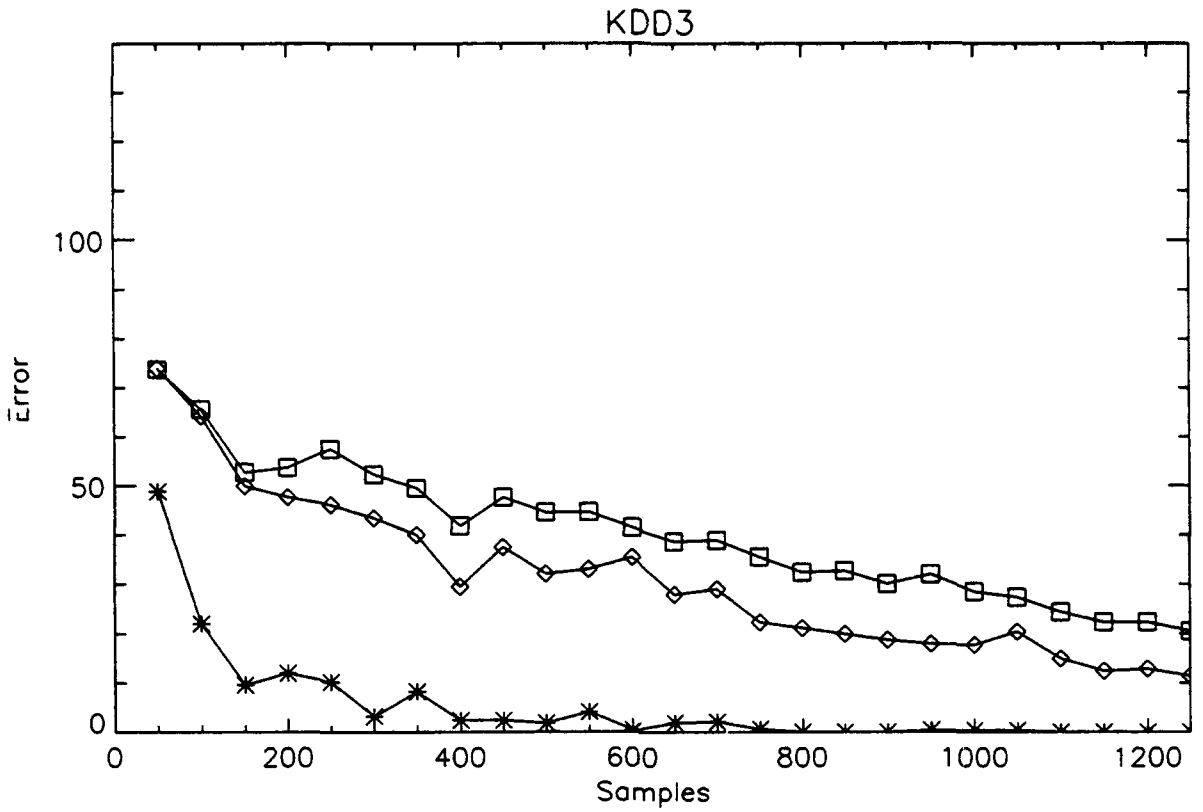
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 −t10 −m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

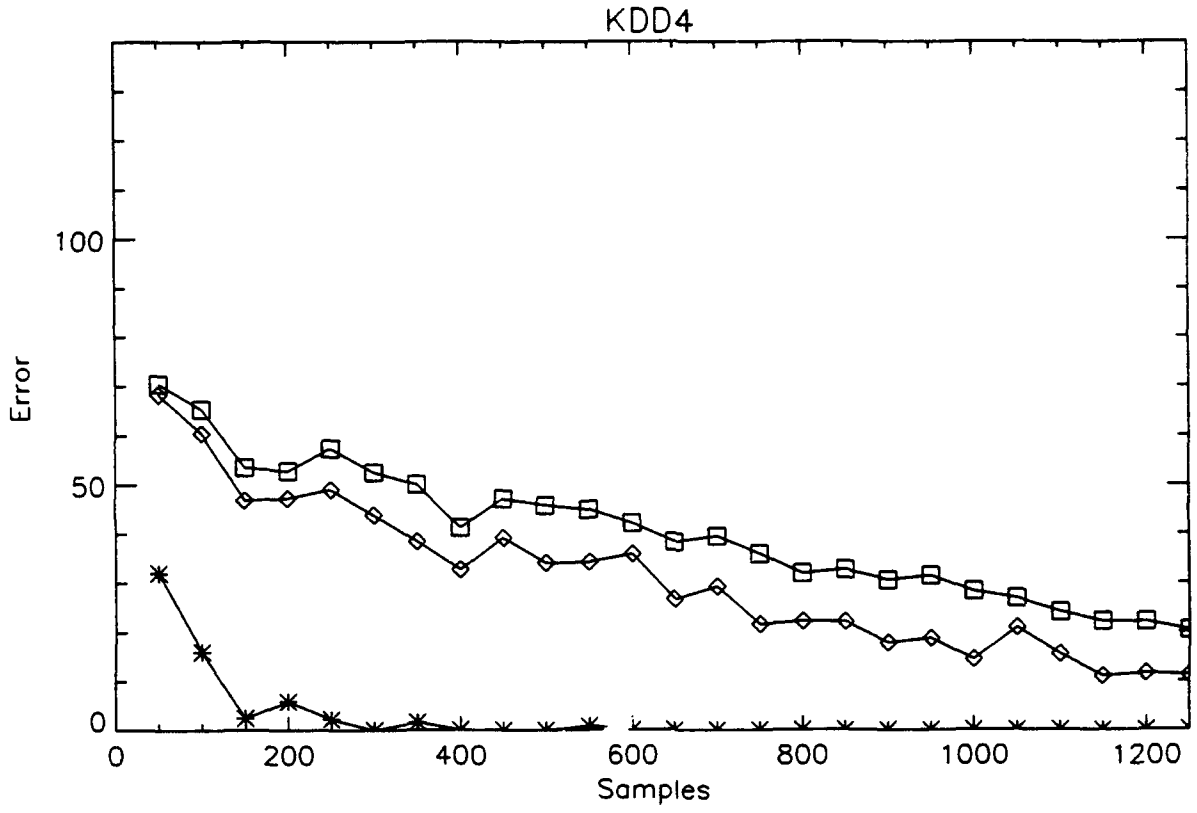
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 -t10 -m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

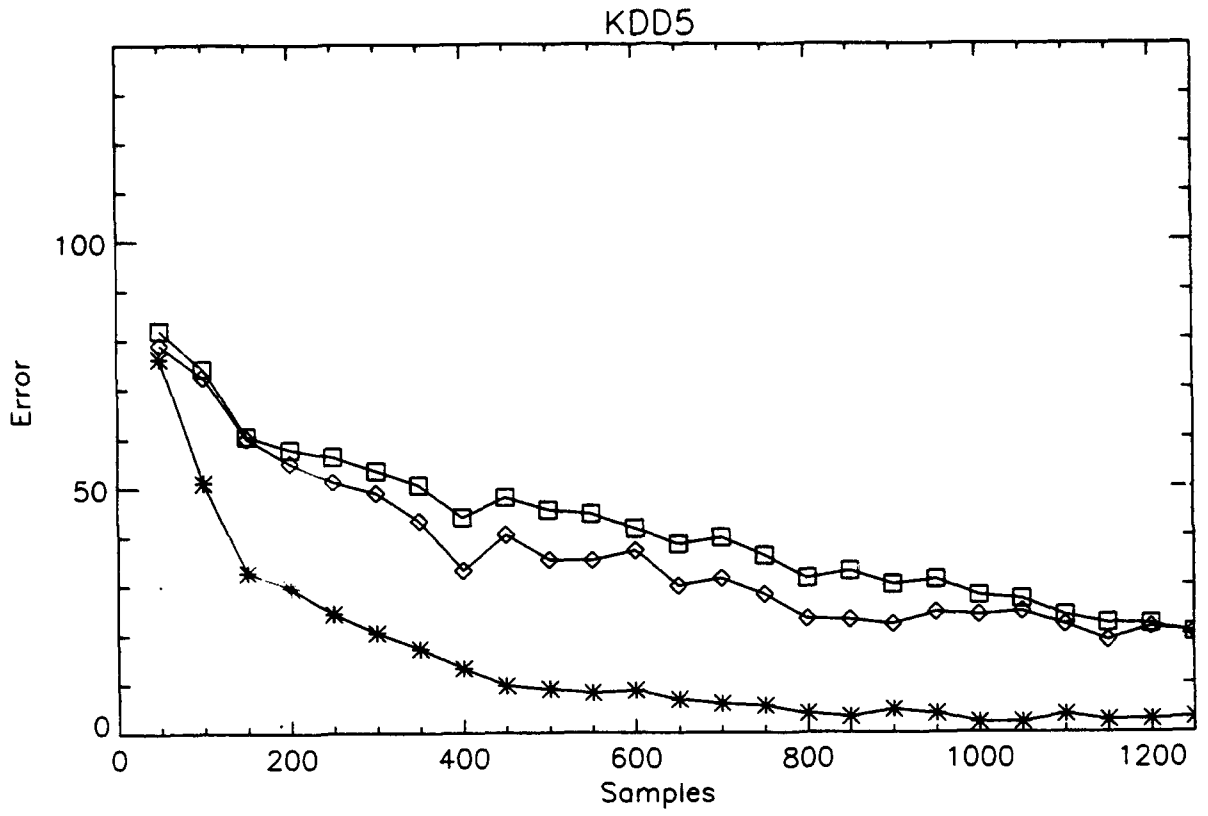
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 −t10 −m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

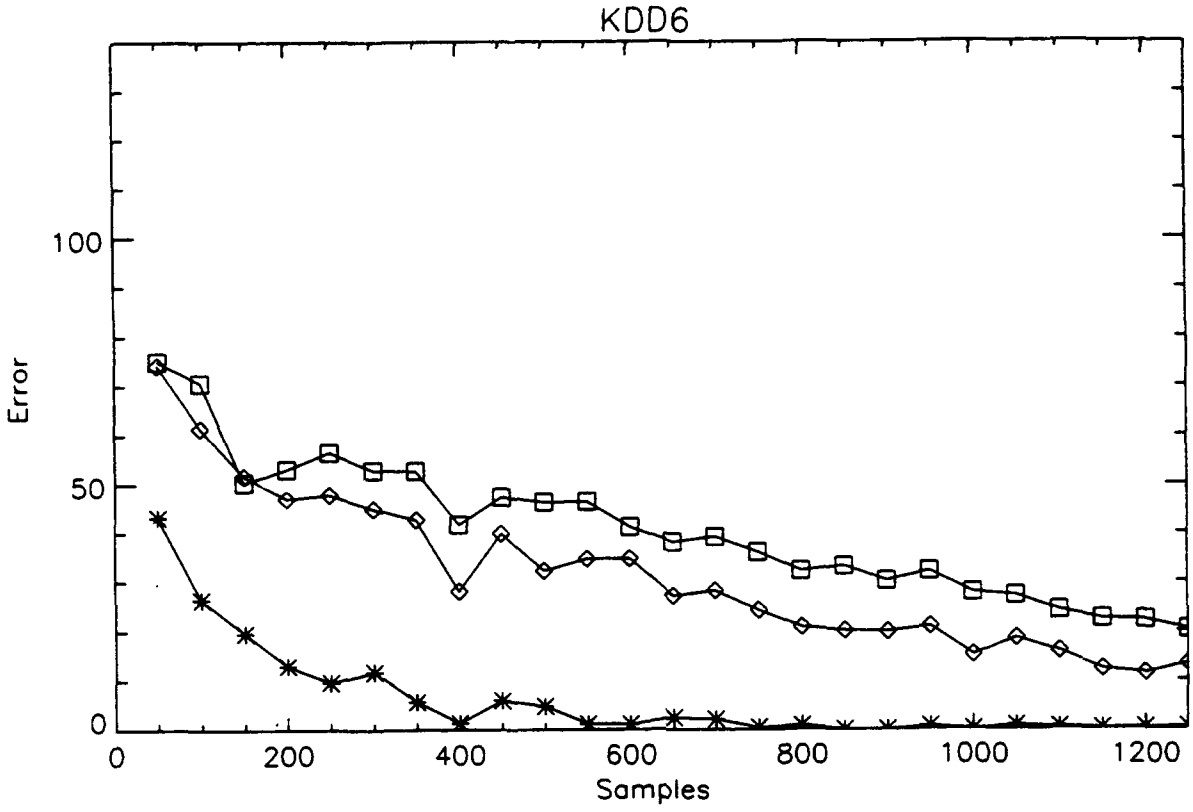
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 -t10 -m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

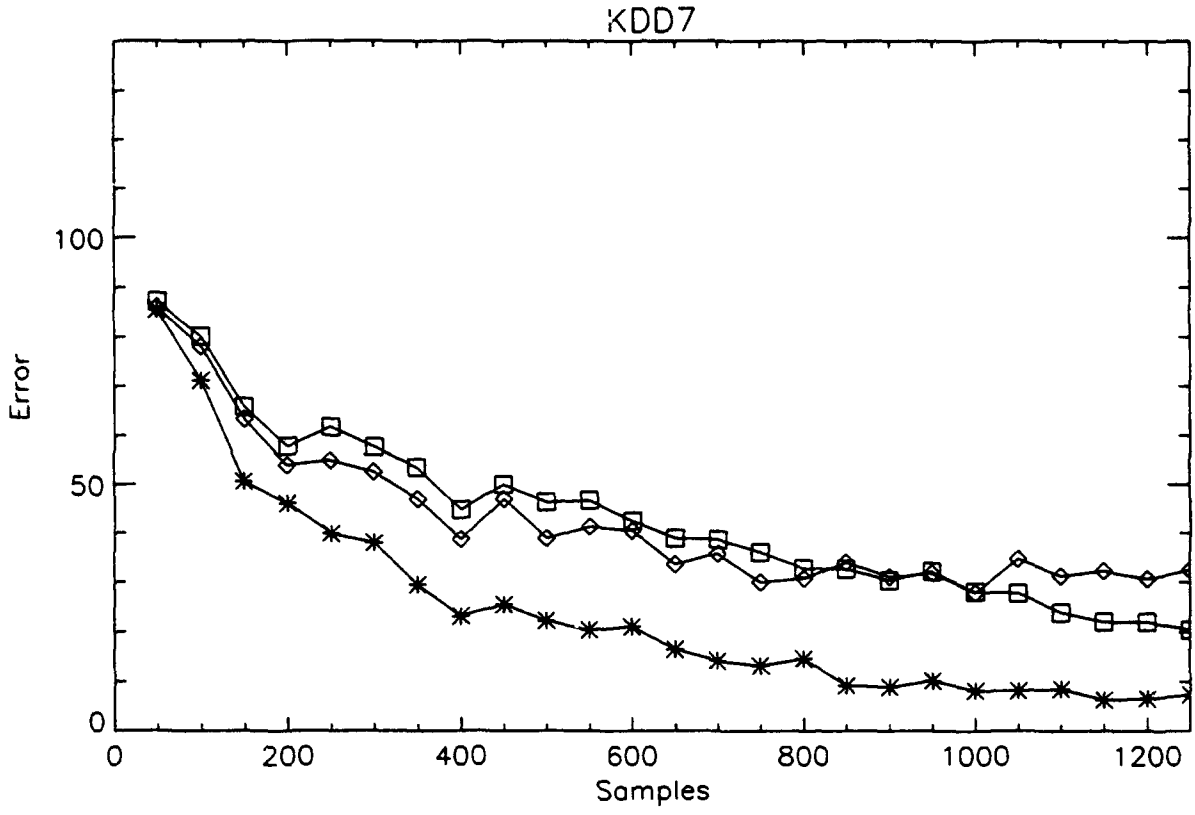
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 -t10 -m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

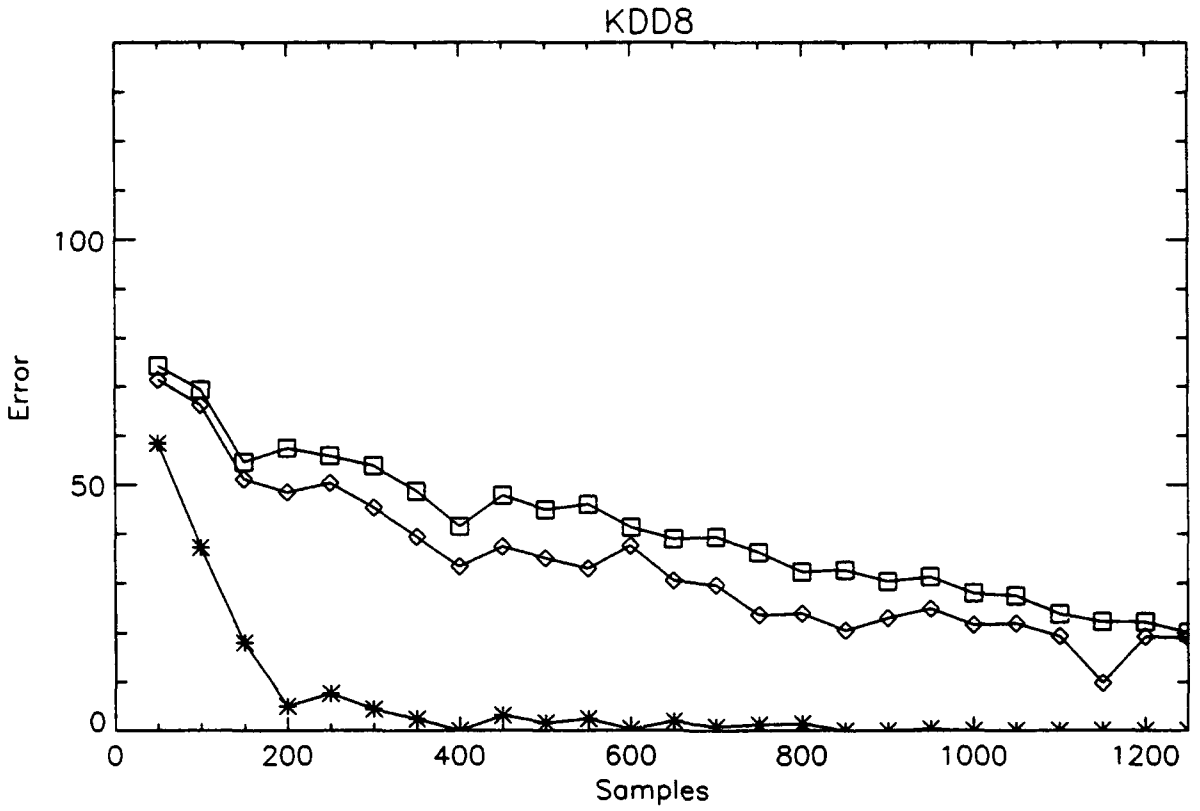
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 -t10 -m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

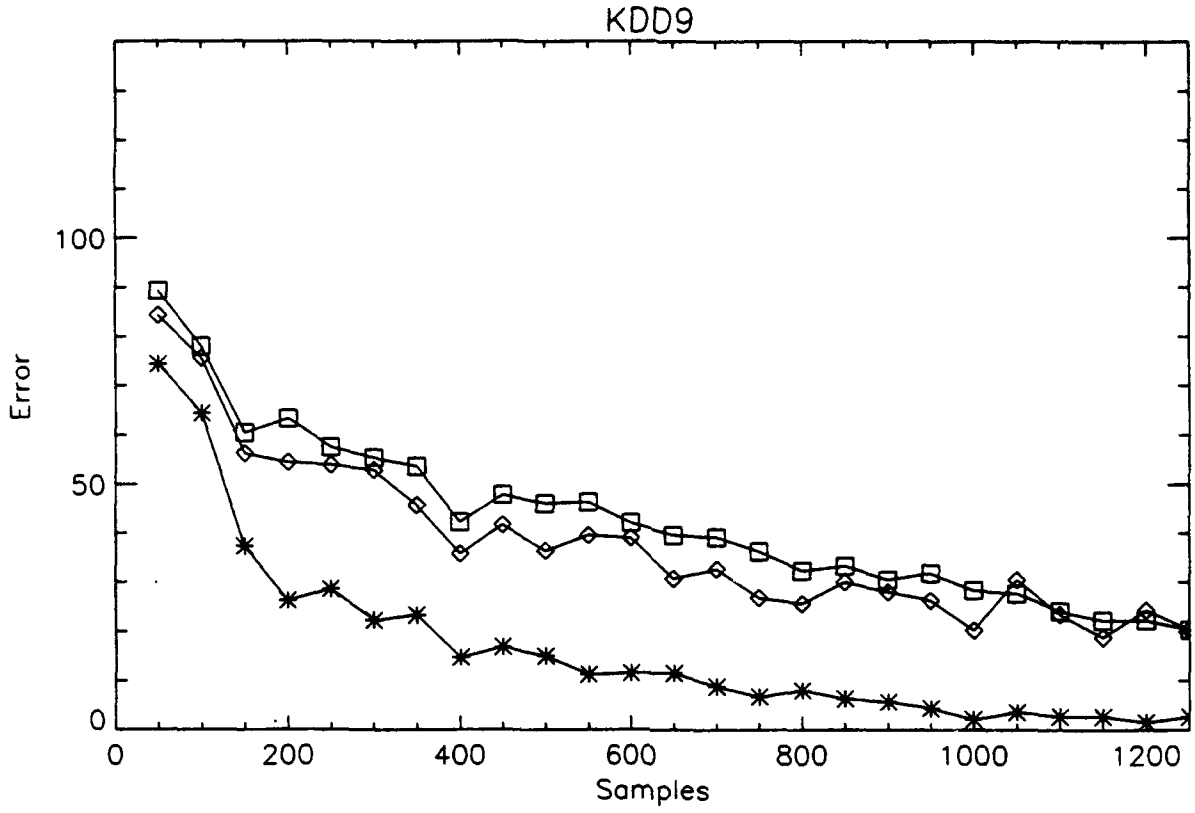
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 −t10 −m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

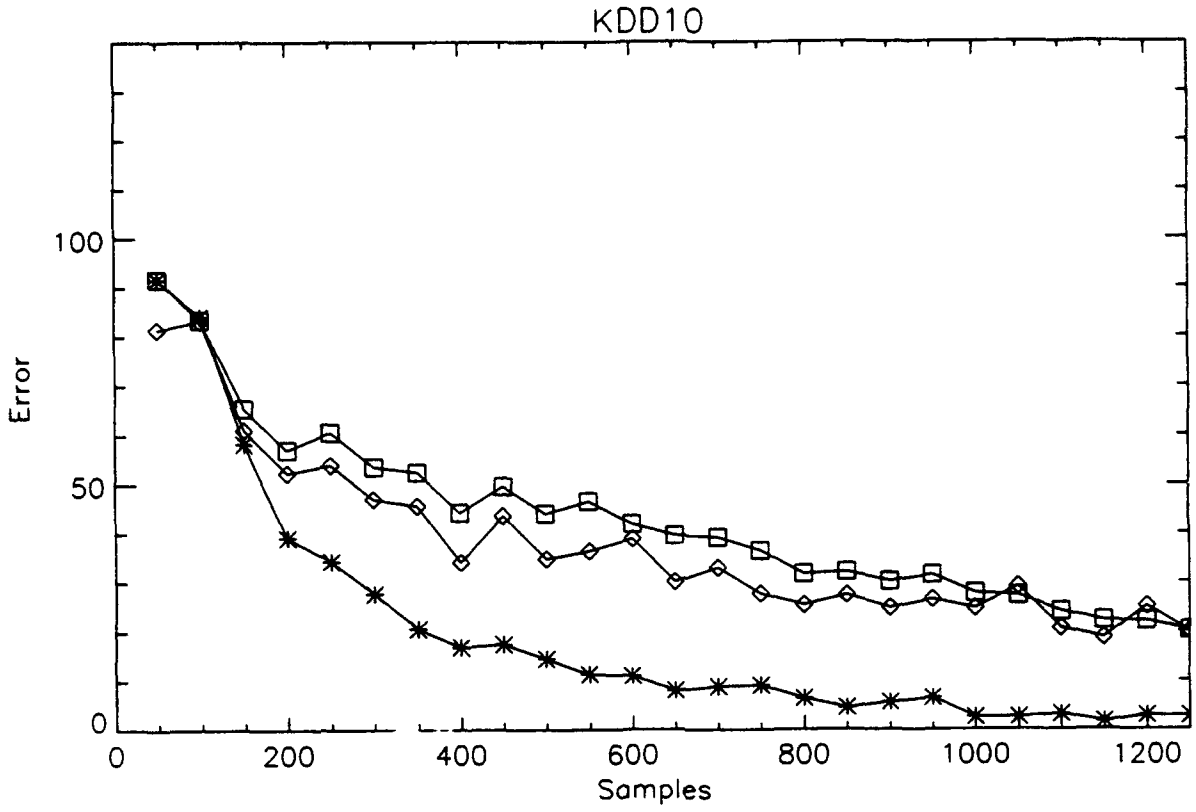
Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

C45 −t10 −m0

with replacement

20% noise



Comparison of preprocessing methods

No Preprocessing, Pruning (average error)

Lose Conflicts, no pruning (average error)

Majority Rules, no pruning (average error)

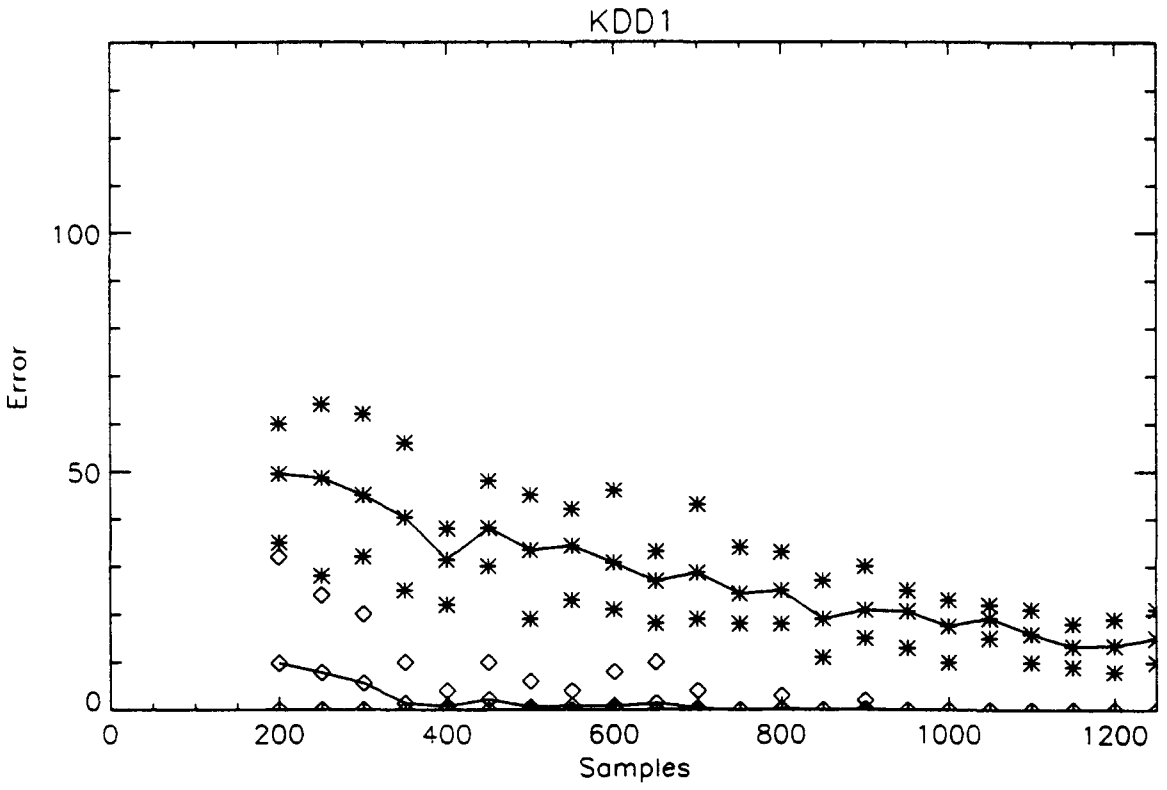
C45 -t10 -m0

with replacement

20% noise

C Additional Graphs with Preprocessing and Pruning

In Appendix C, there are some additional graphs of the 10 functions. The set shows the 10 functions with the 3 different preprocessing methods, however, we compare pruned and unpruned for each of the preprocessing methods. Again, we mention that it did not make sense to prune the tree when no conflicts were given (preprocessing) if we are trying to compare preprocessing with internal noise handling. However, the tests were performed before this realisation was made.



Chance

Max error no prunning

Min error no prunning

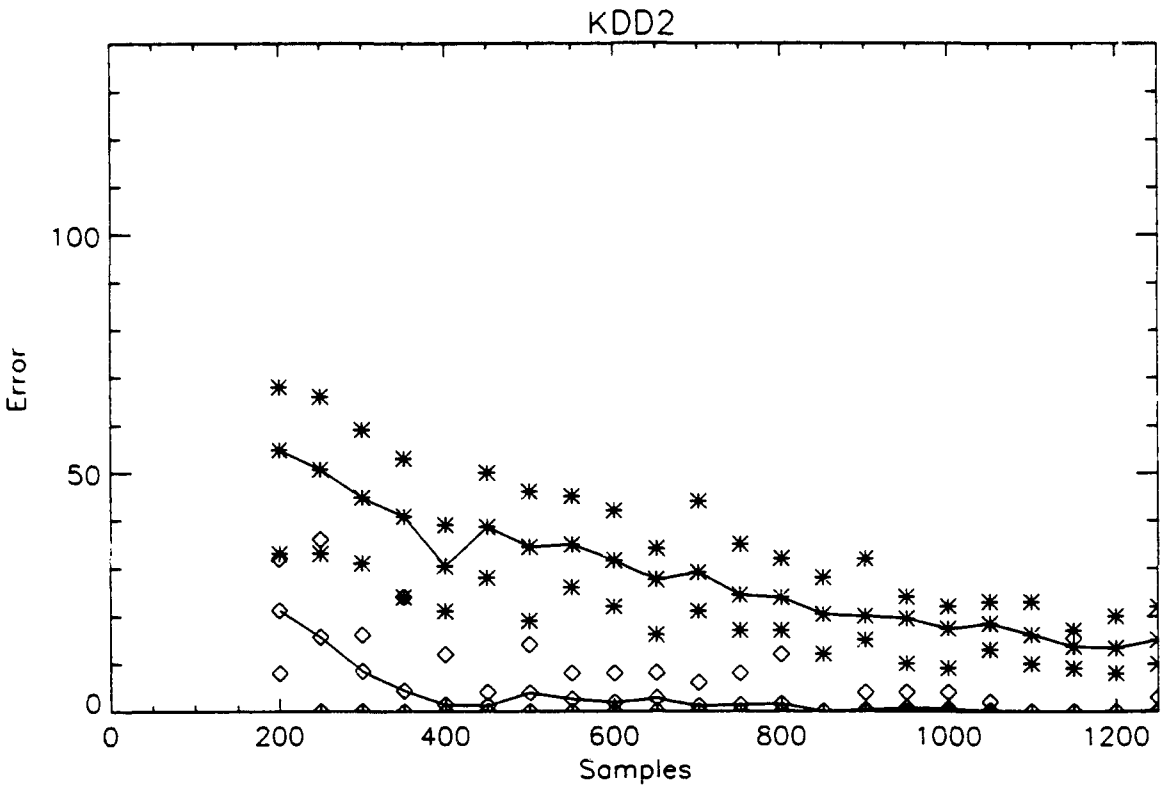
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise−no preprocessing



Chance

Max error no prunning

Min error no prunning

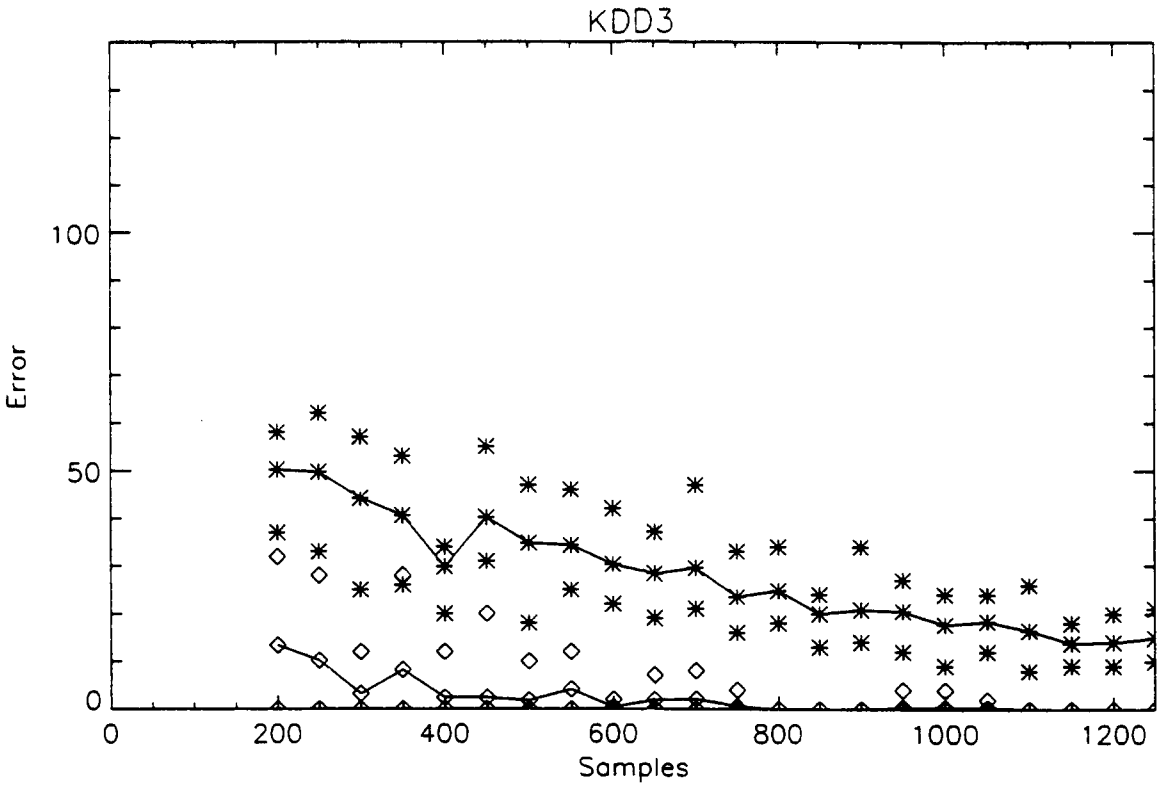
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

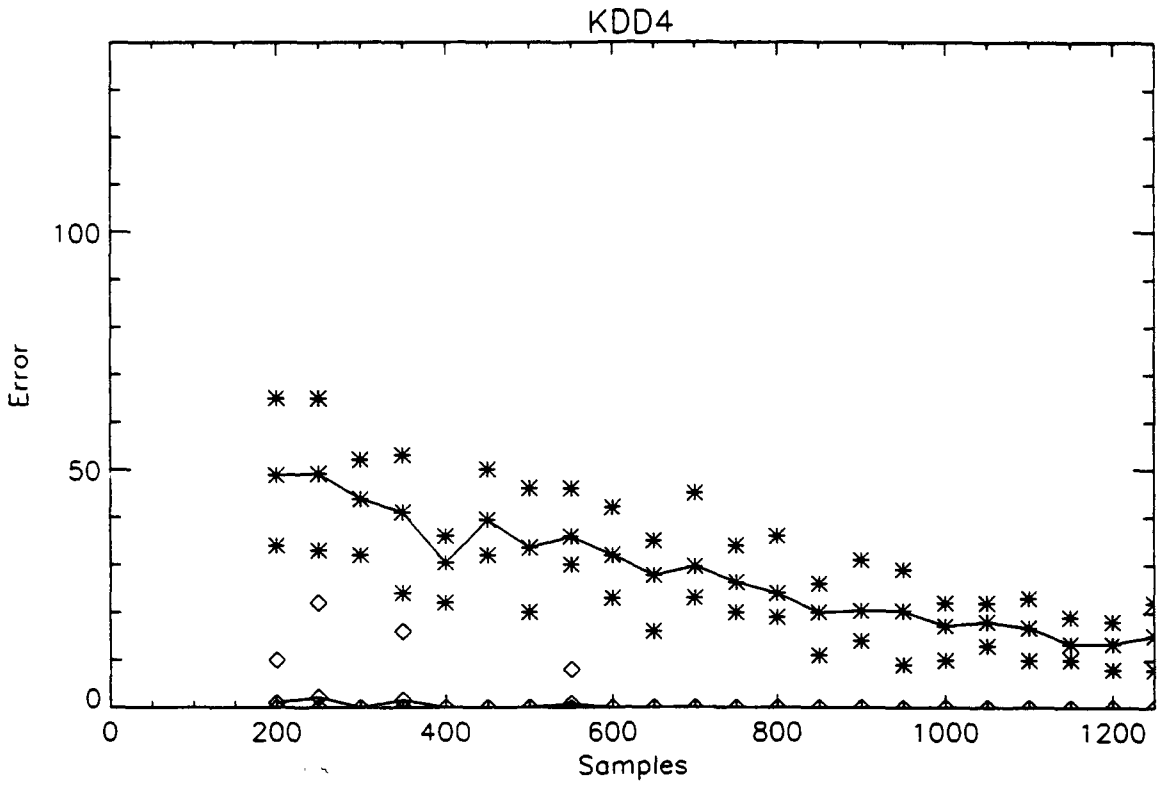
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

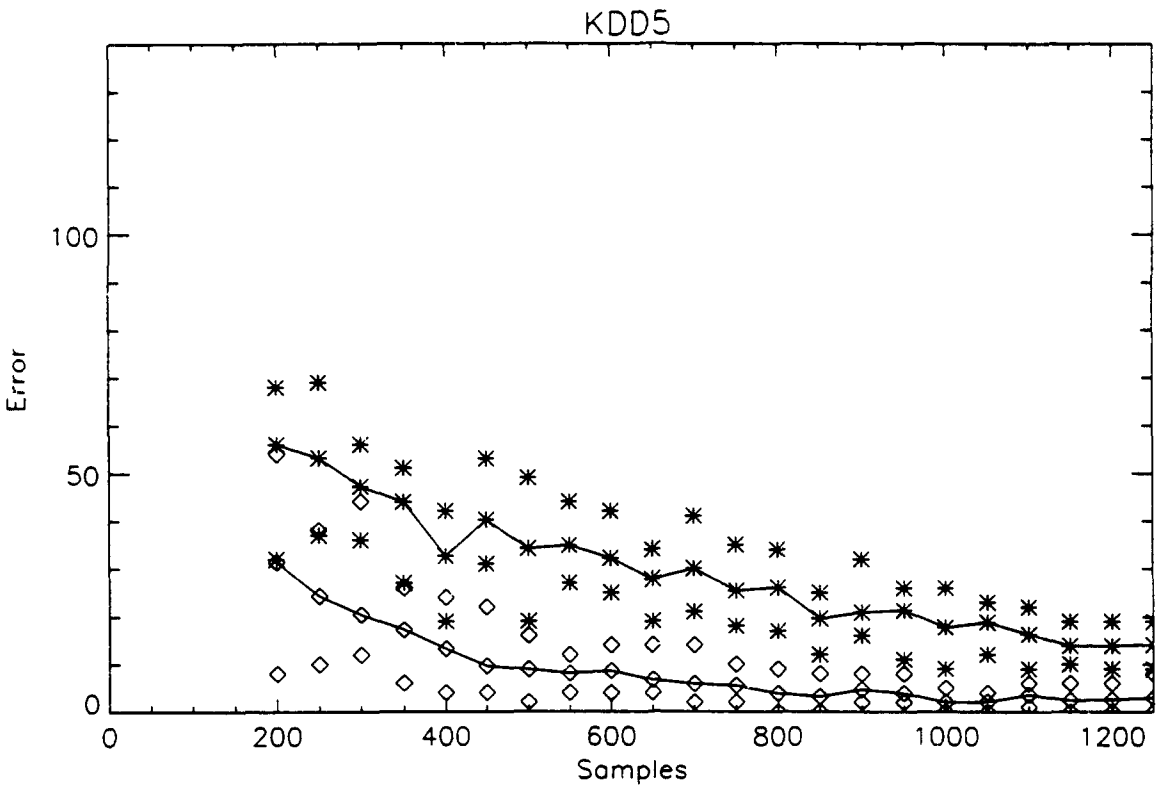
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

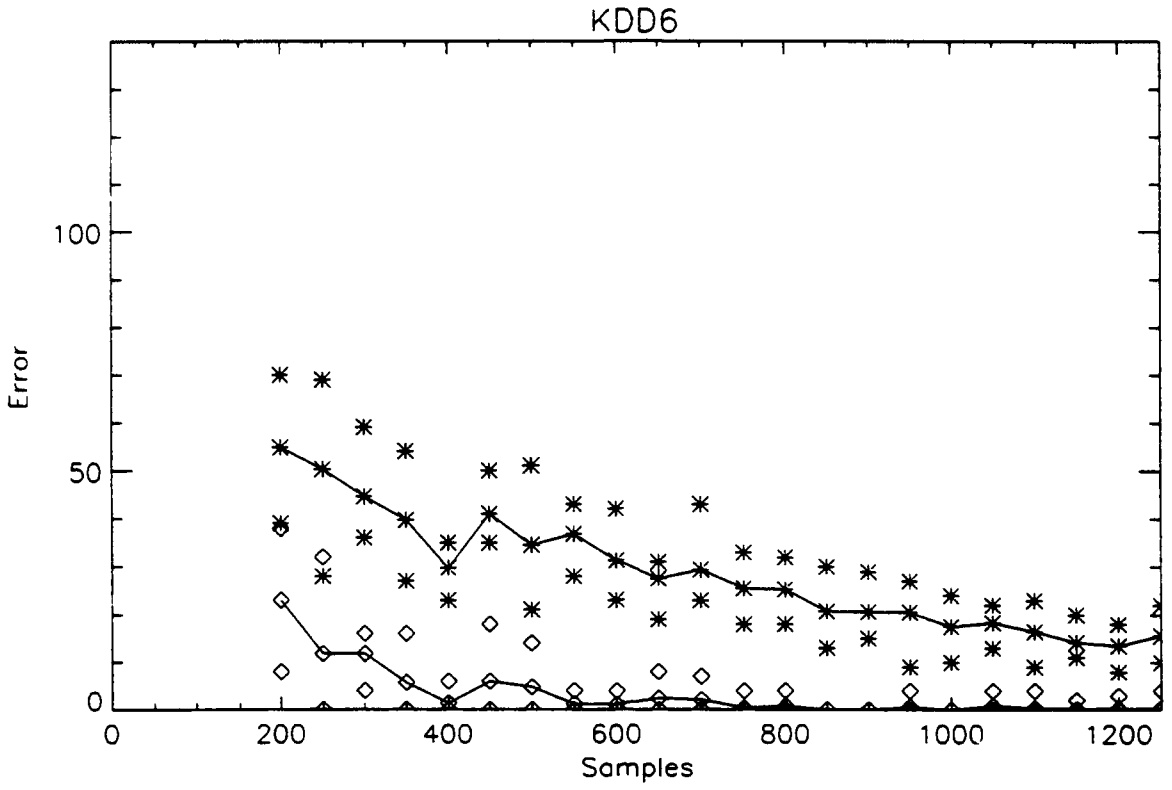
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

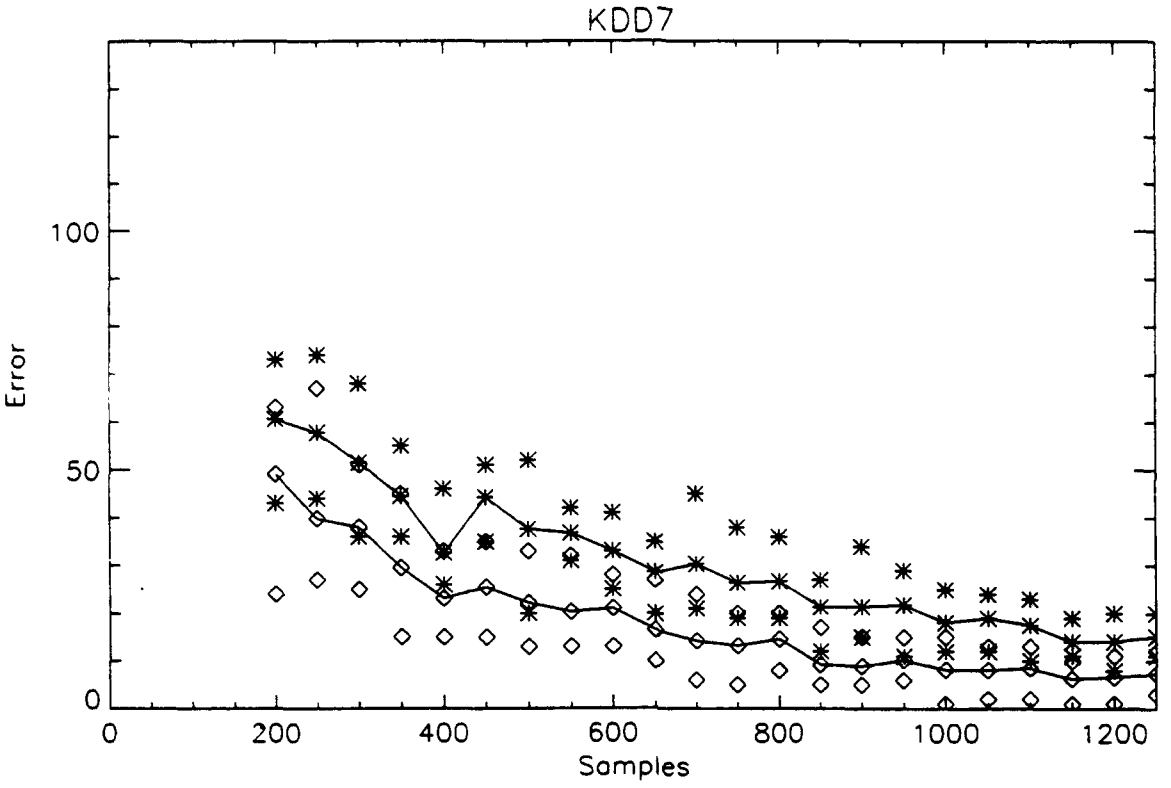
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

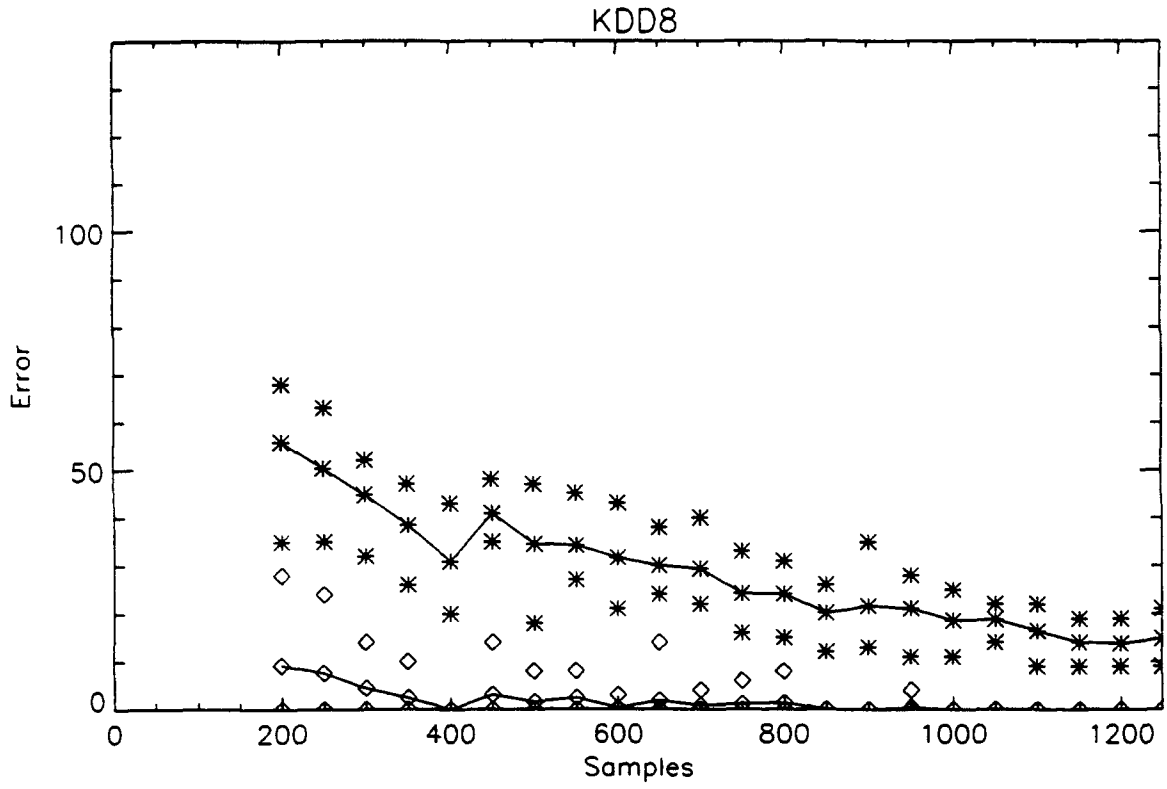
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

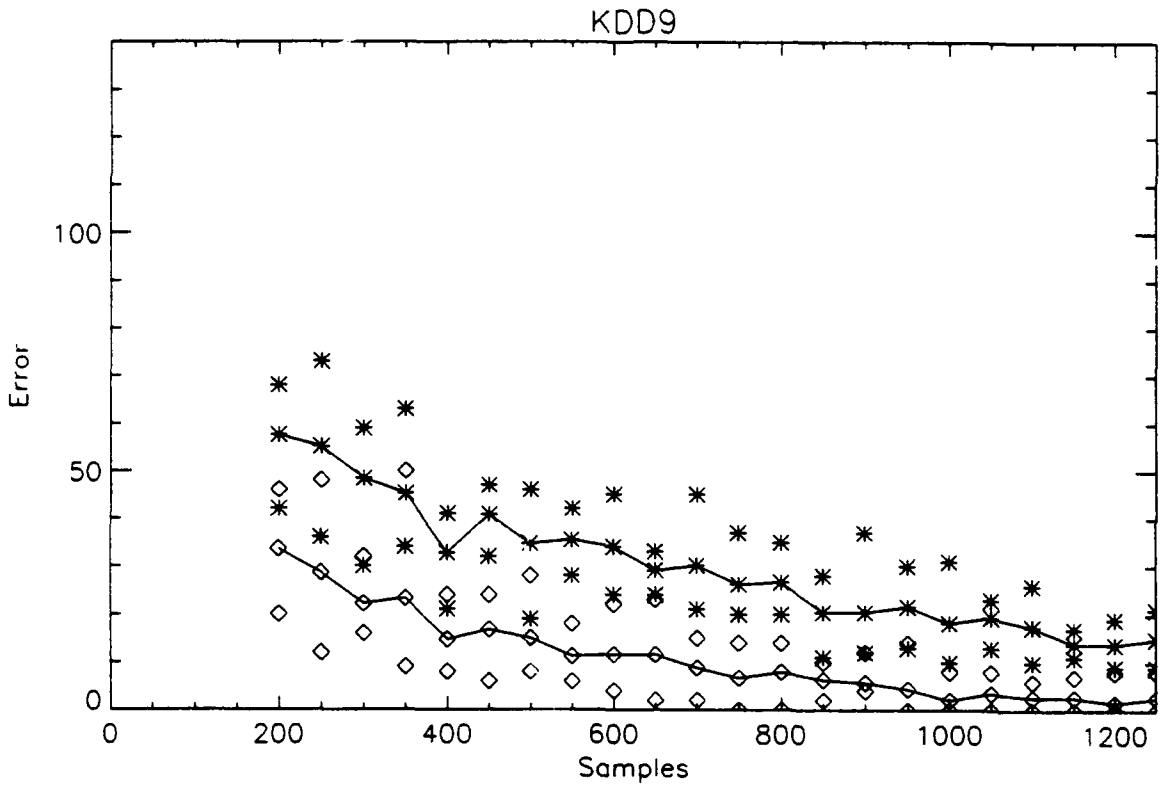
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

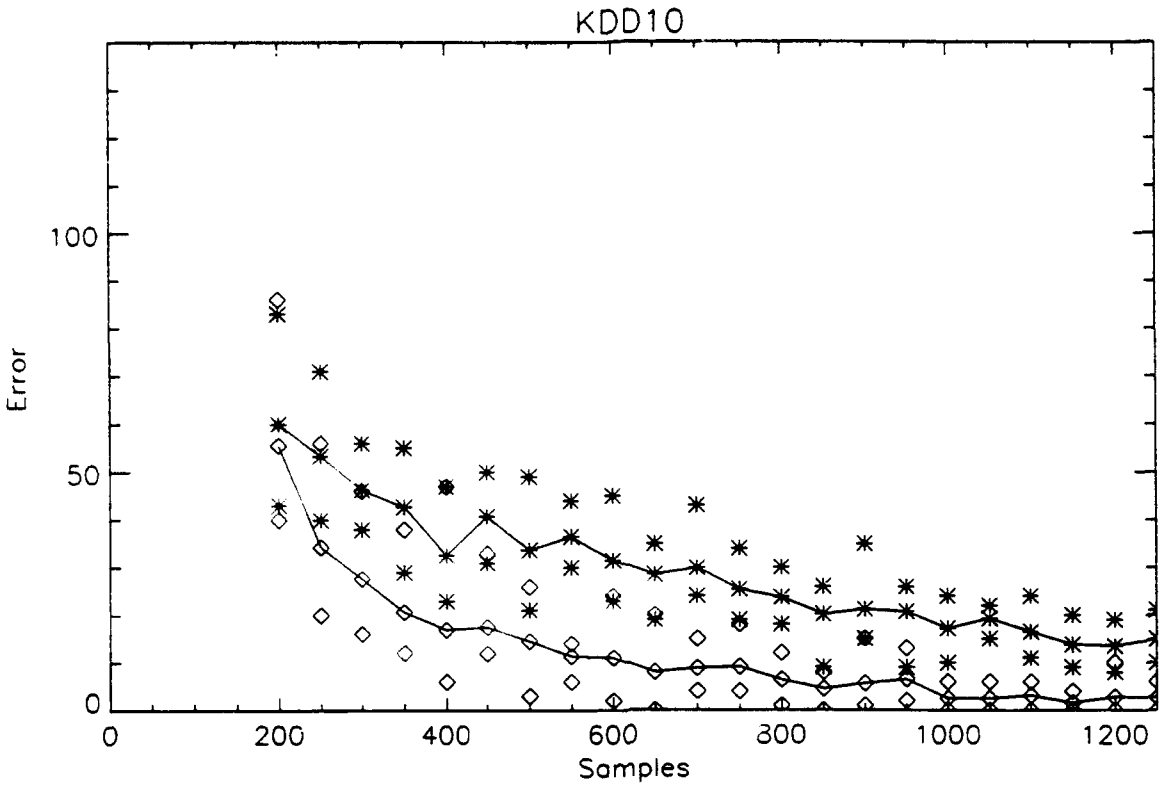
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

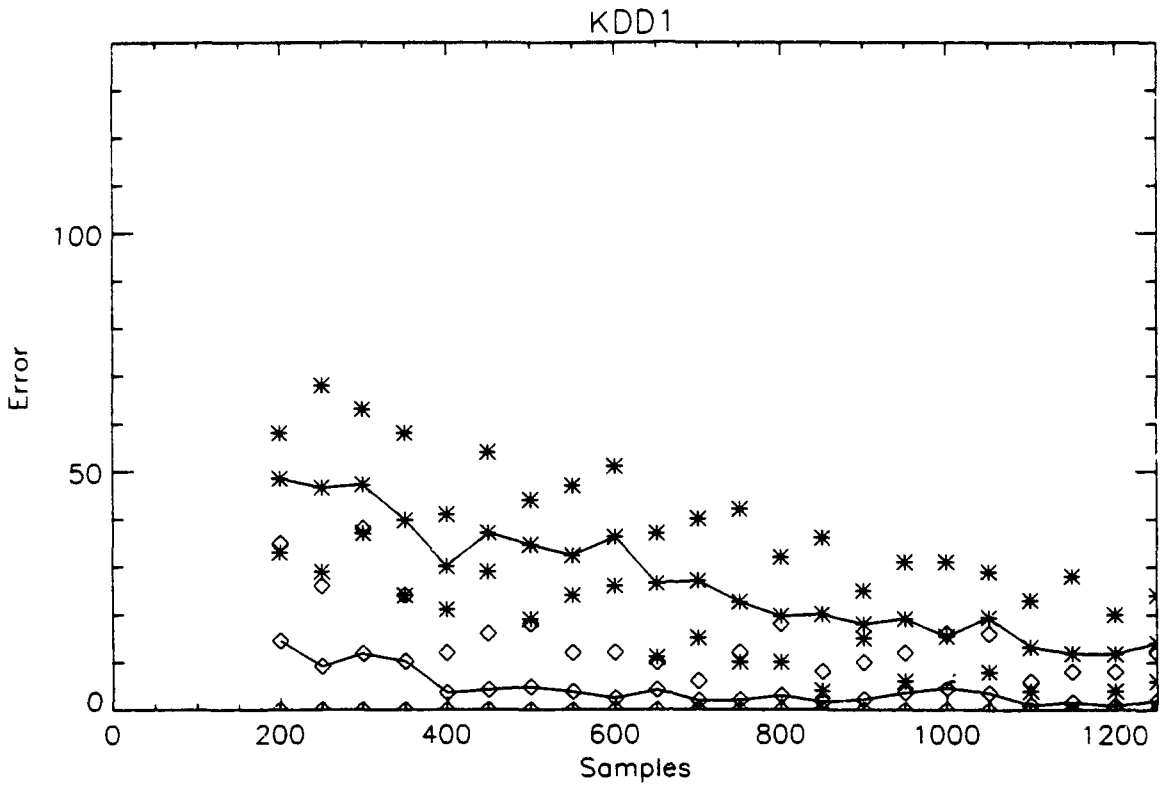
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, no preprocessing



Chance

Max error no prunning

Min error no prunning

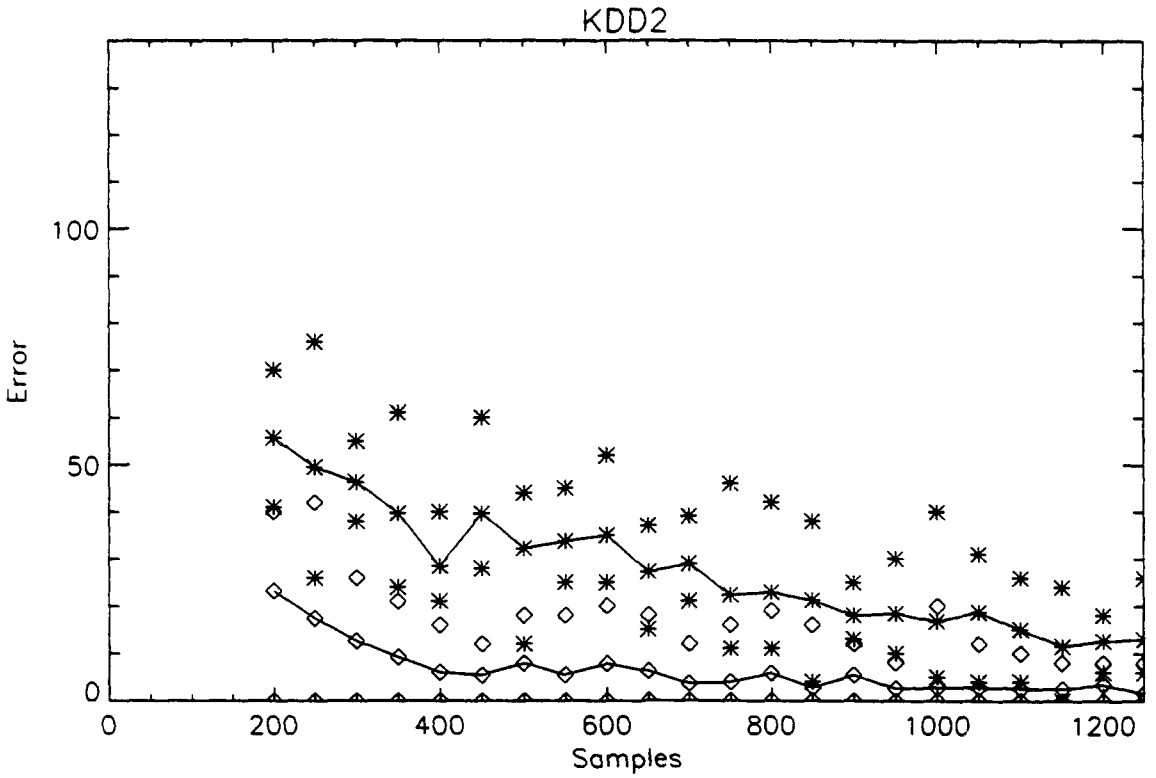
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 -t10 -m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

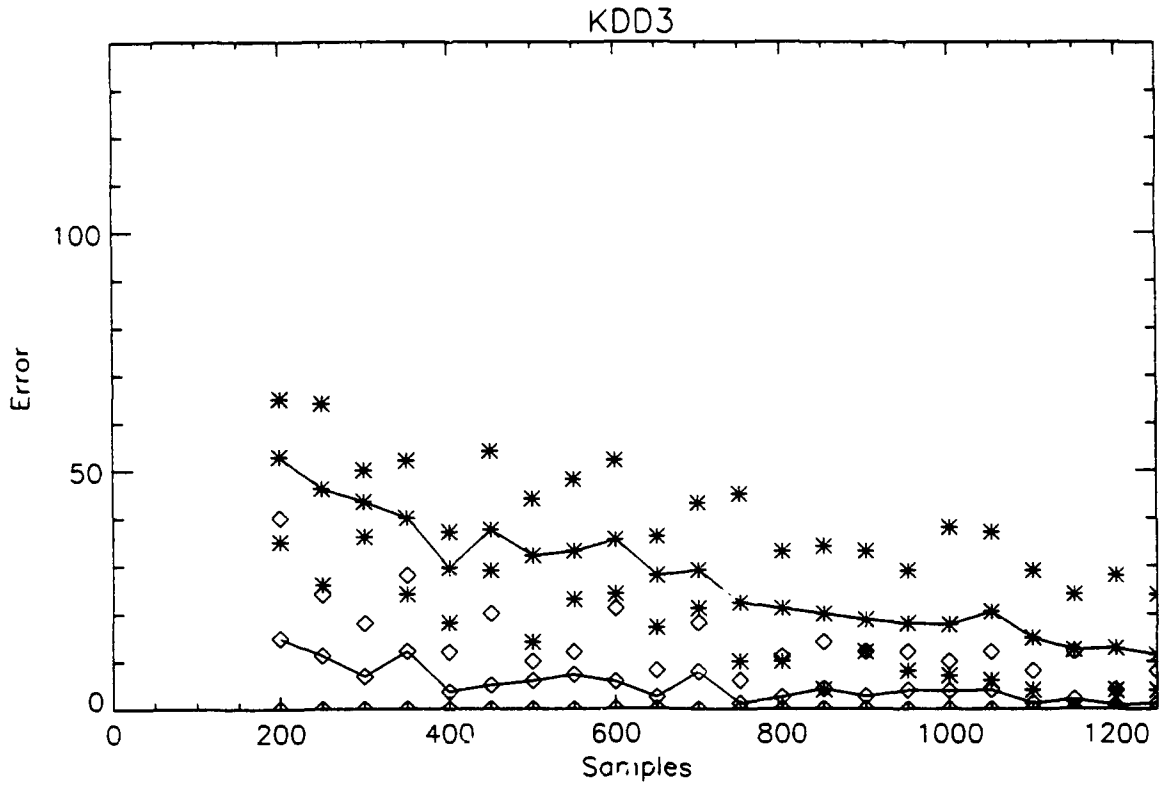
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

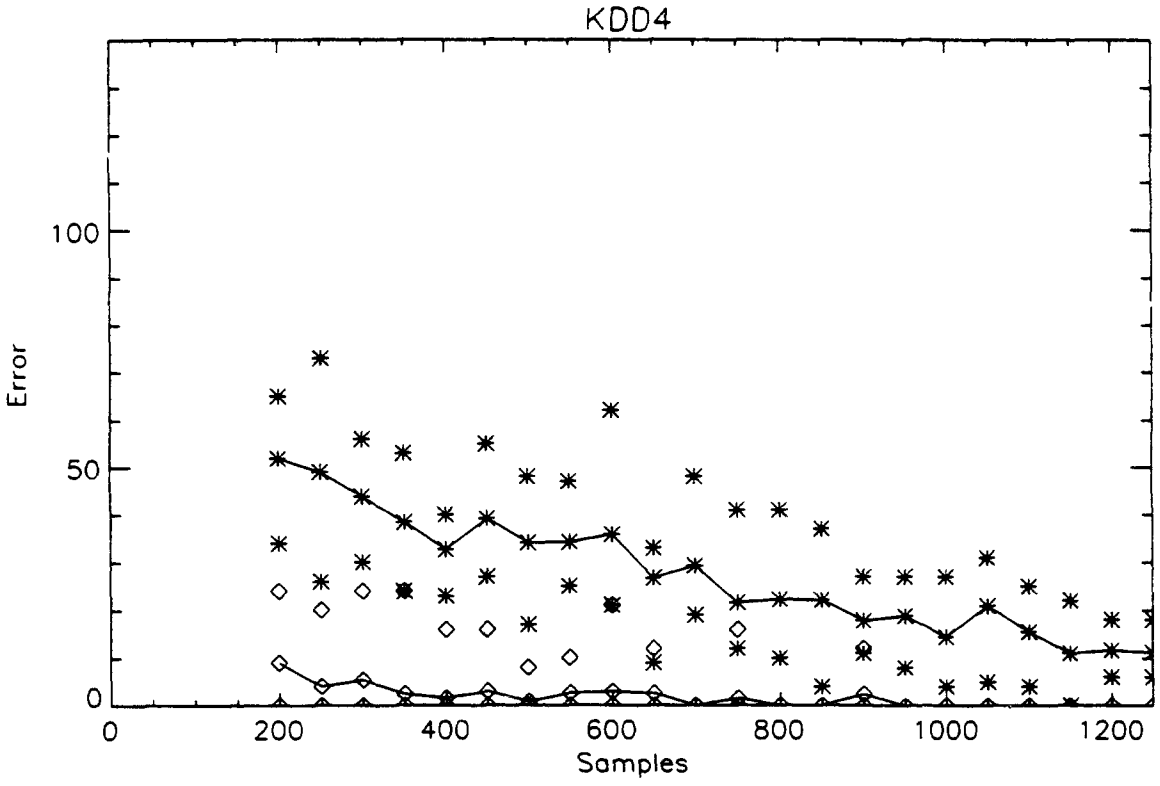
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

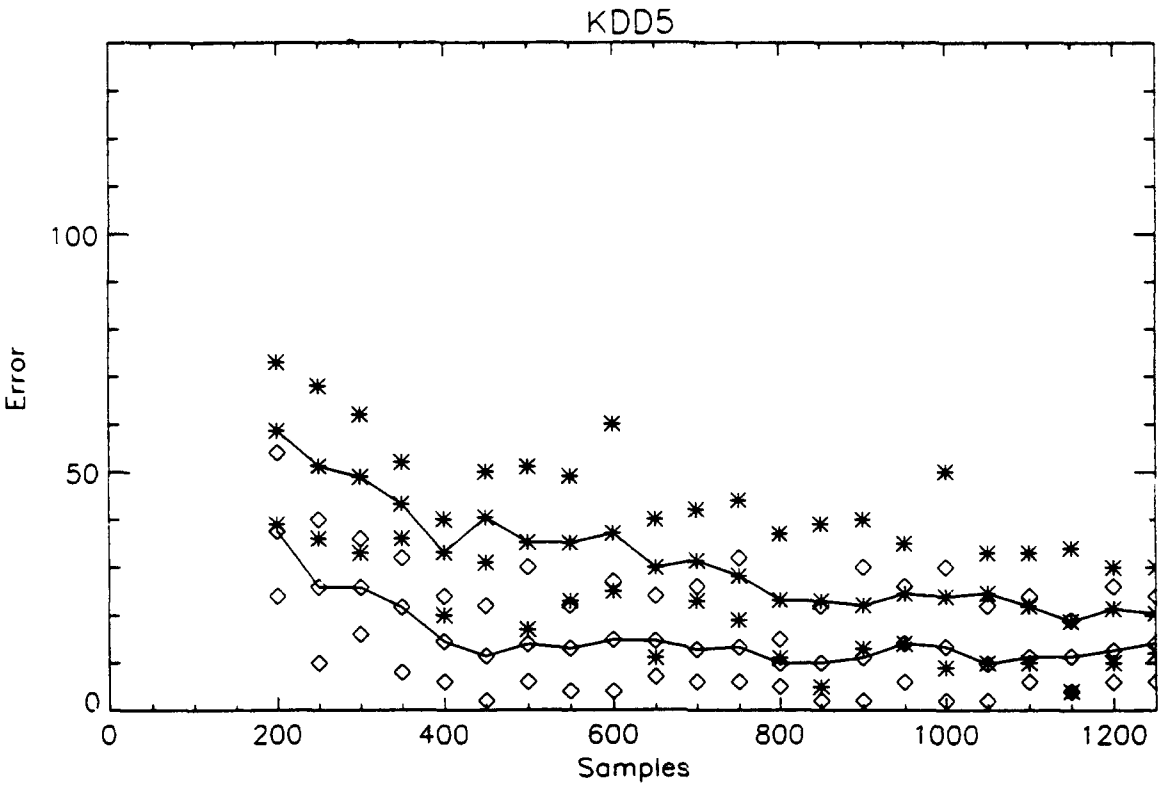
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

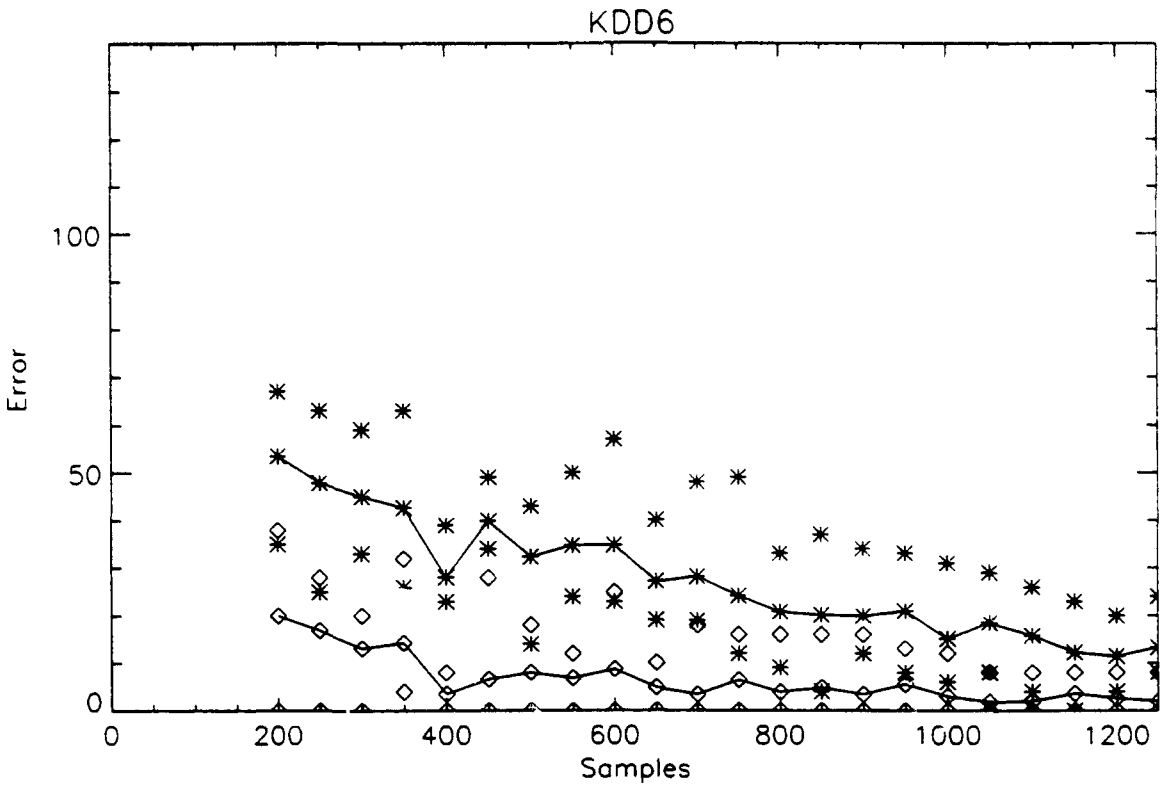
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

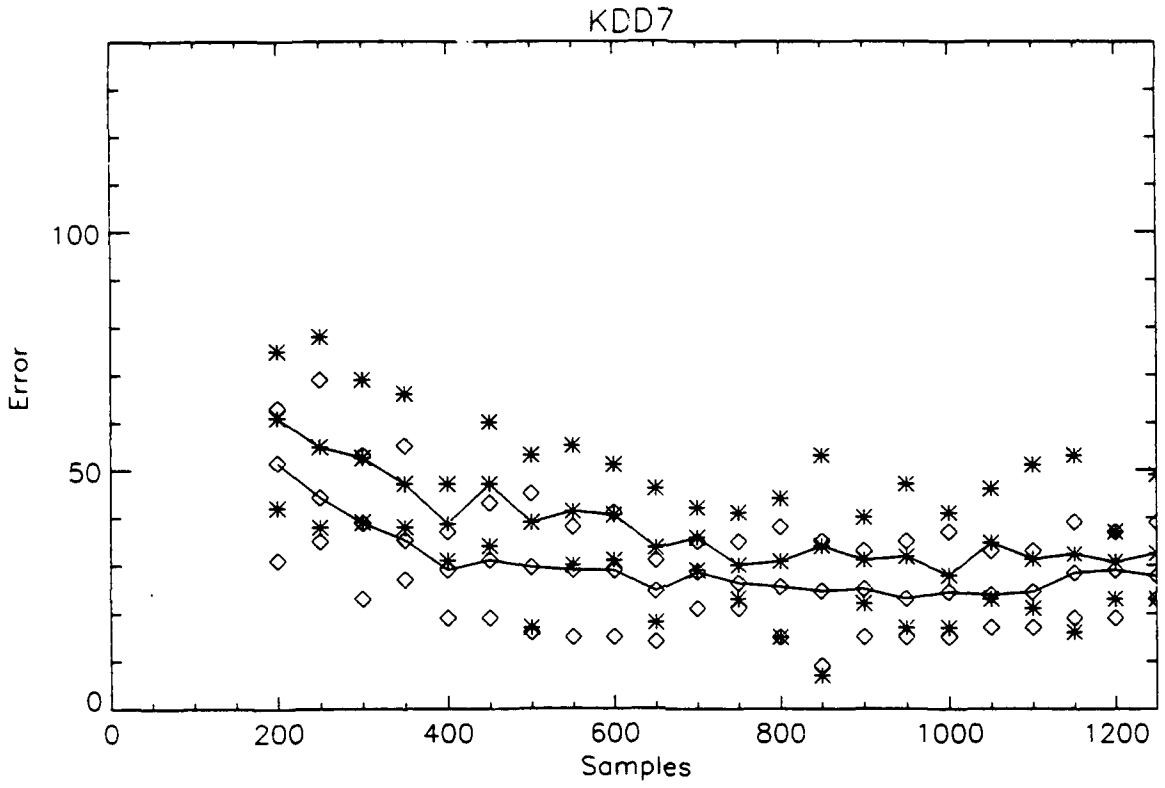
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

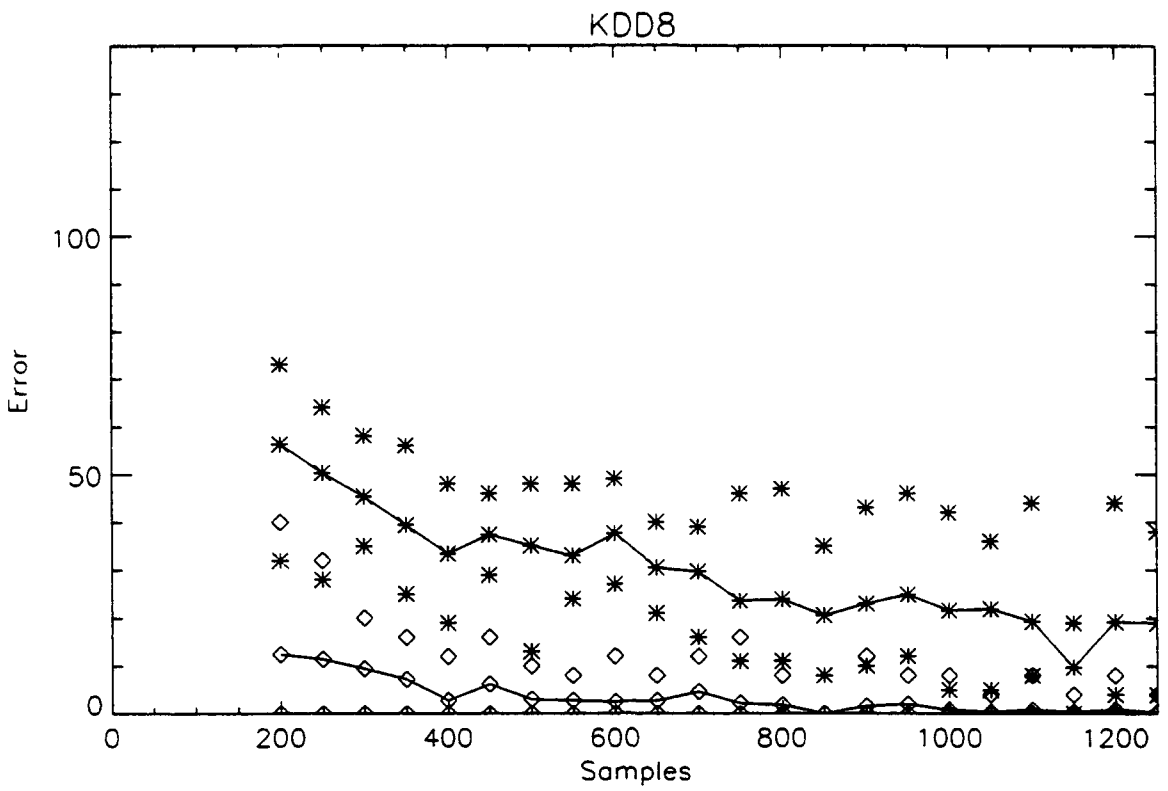
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

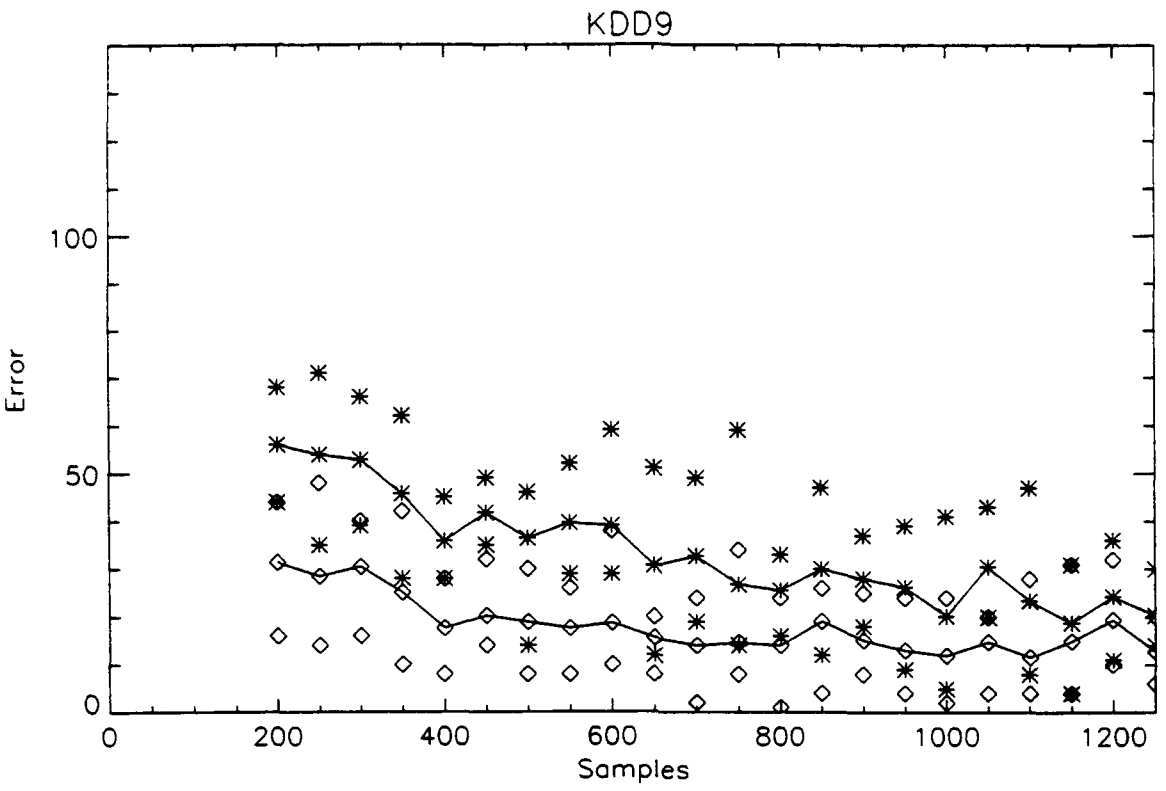
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

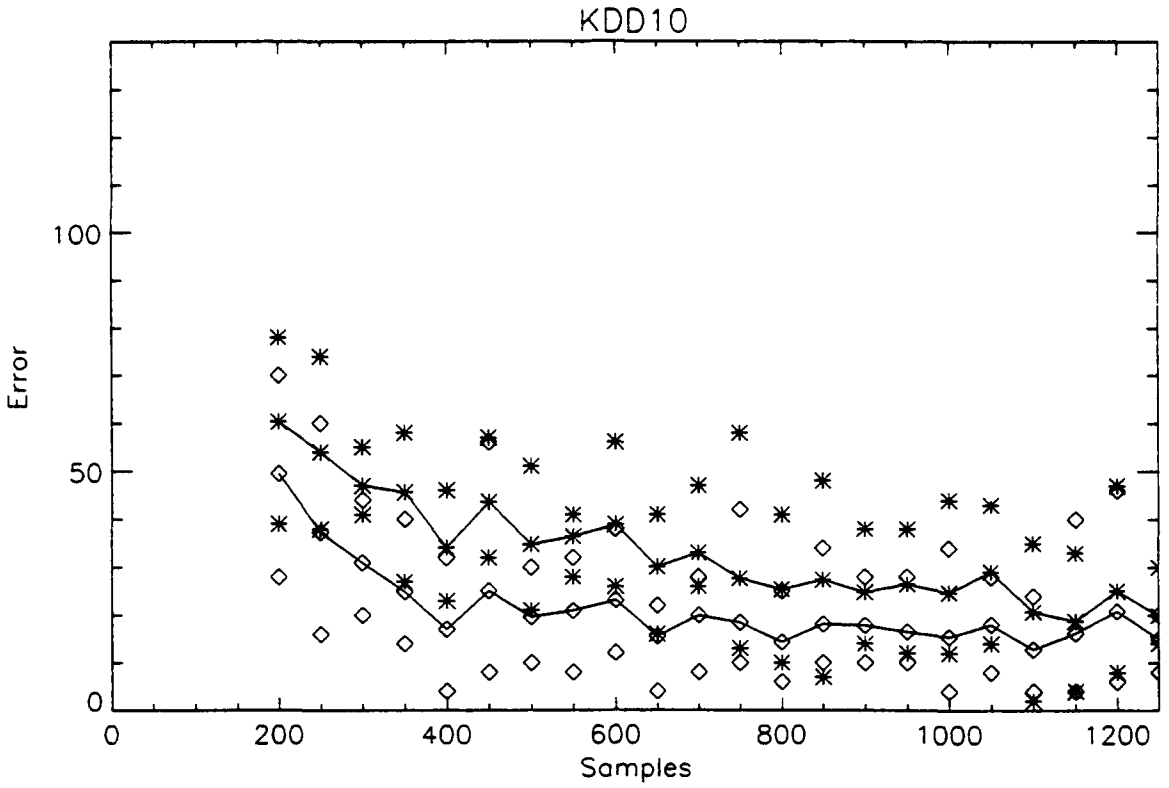
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

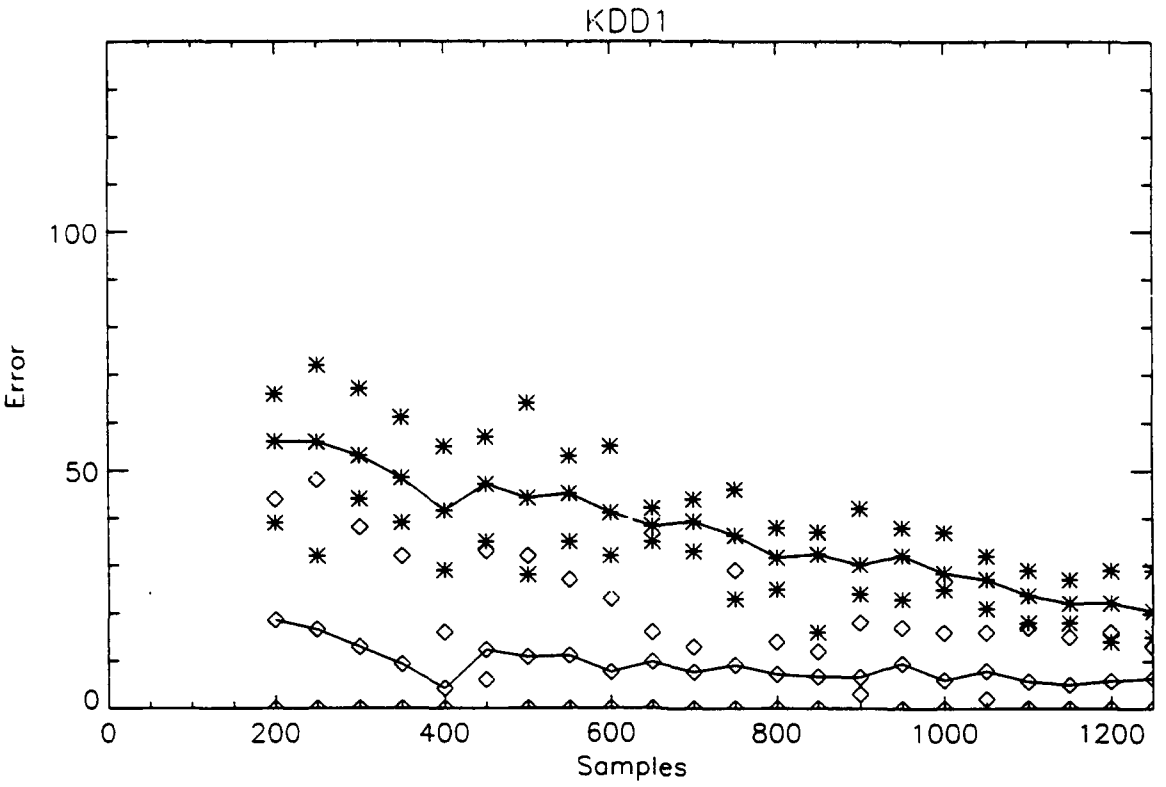
Avg error no prunning

Max error. prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Lose Conflicts



Chance

Max error no prunning

Min error no prunning

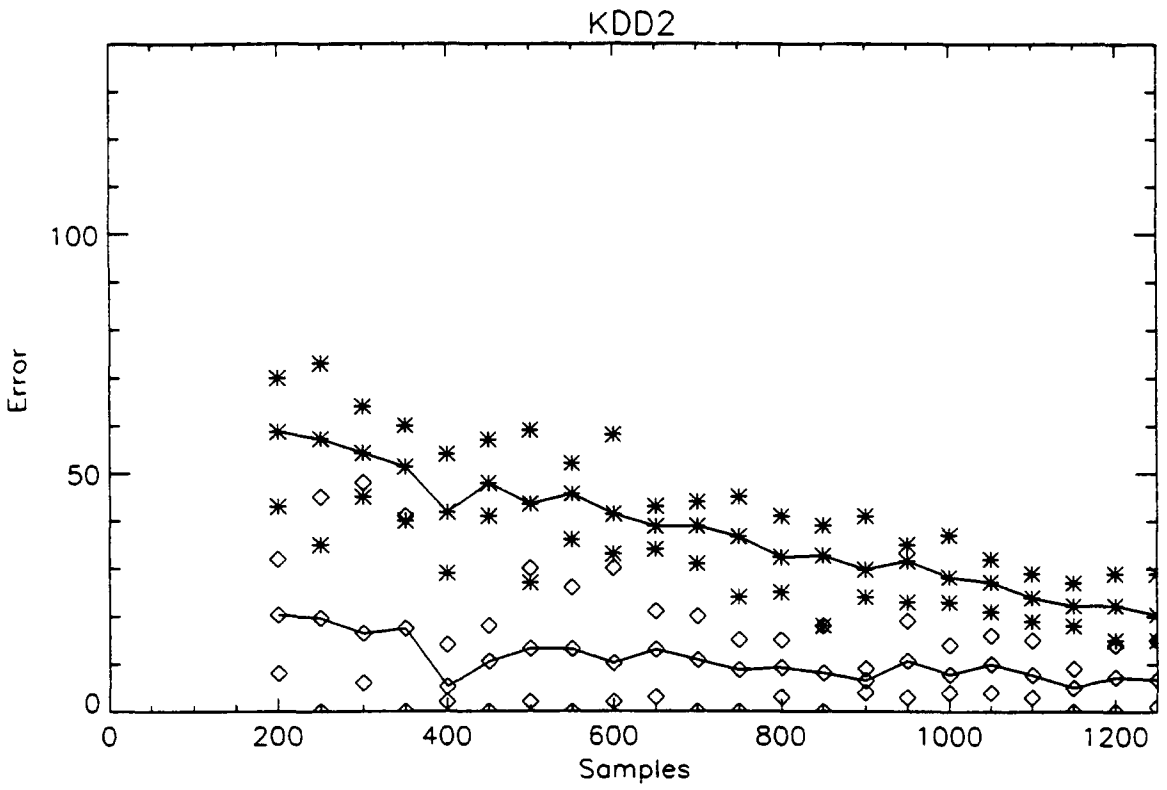
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

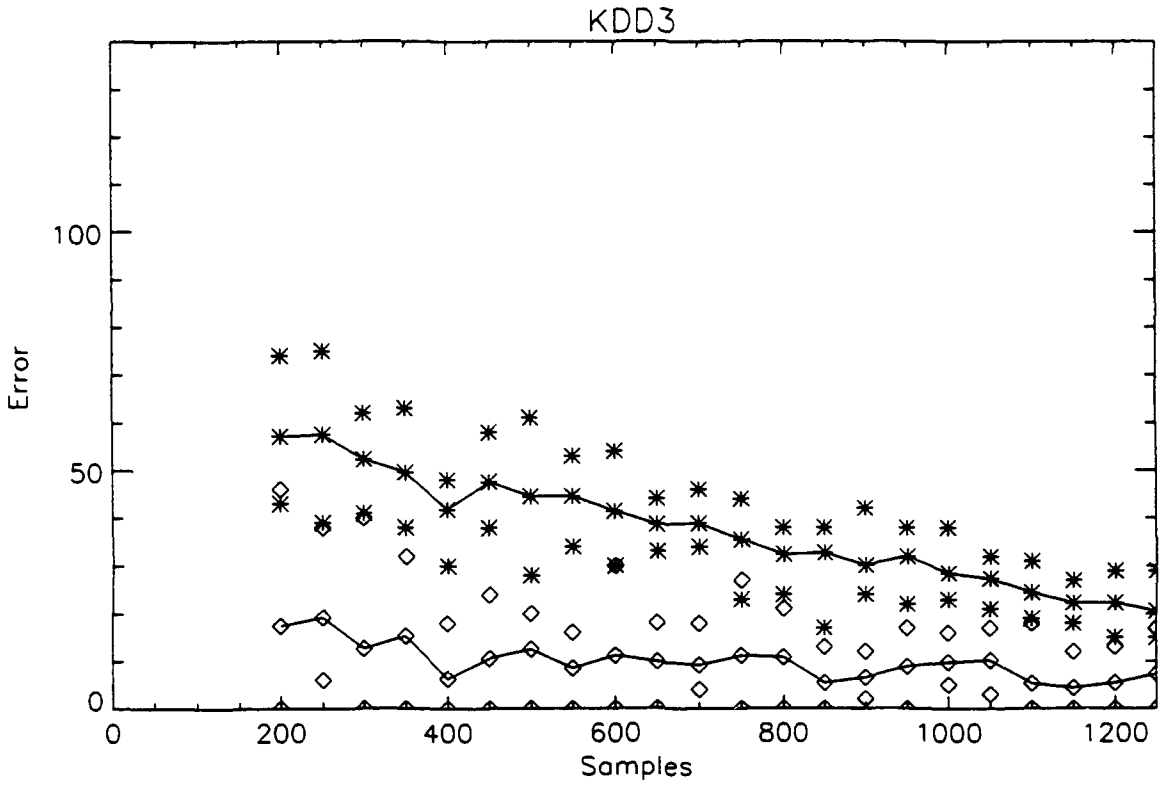
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

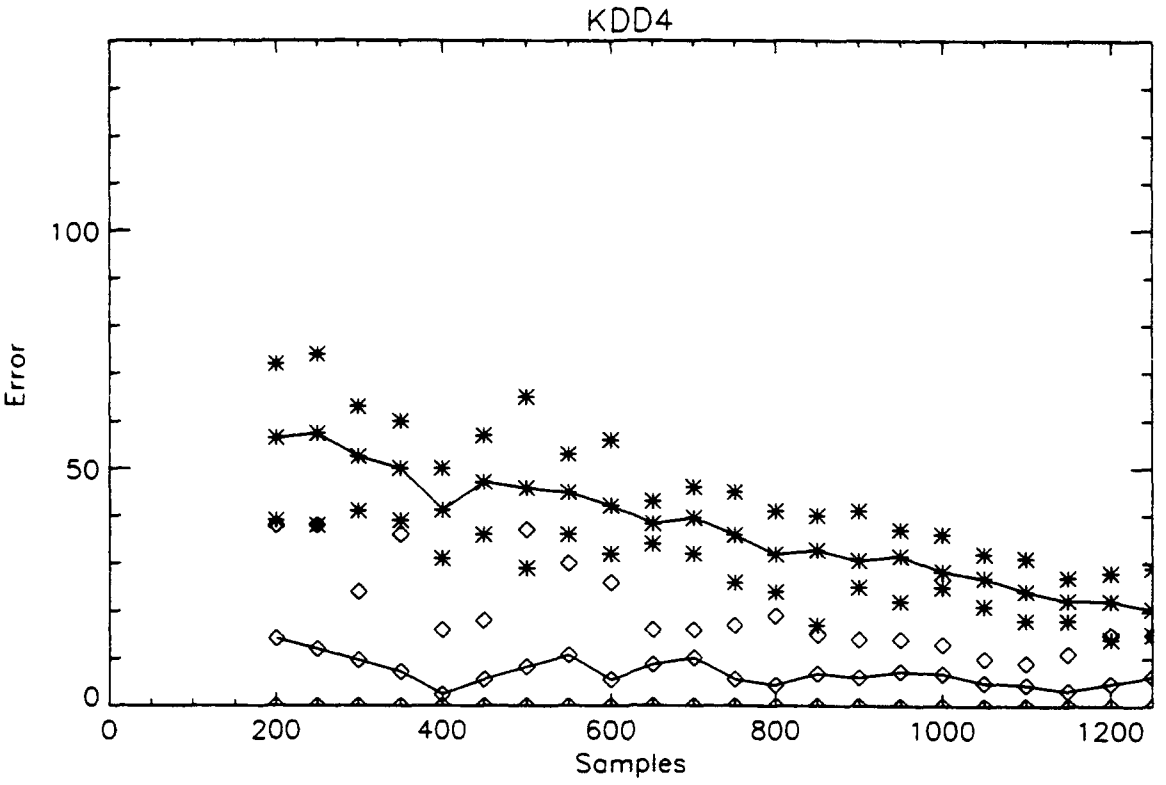
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

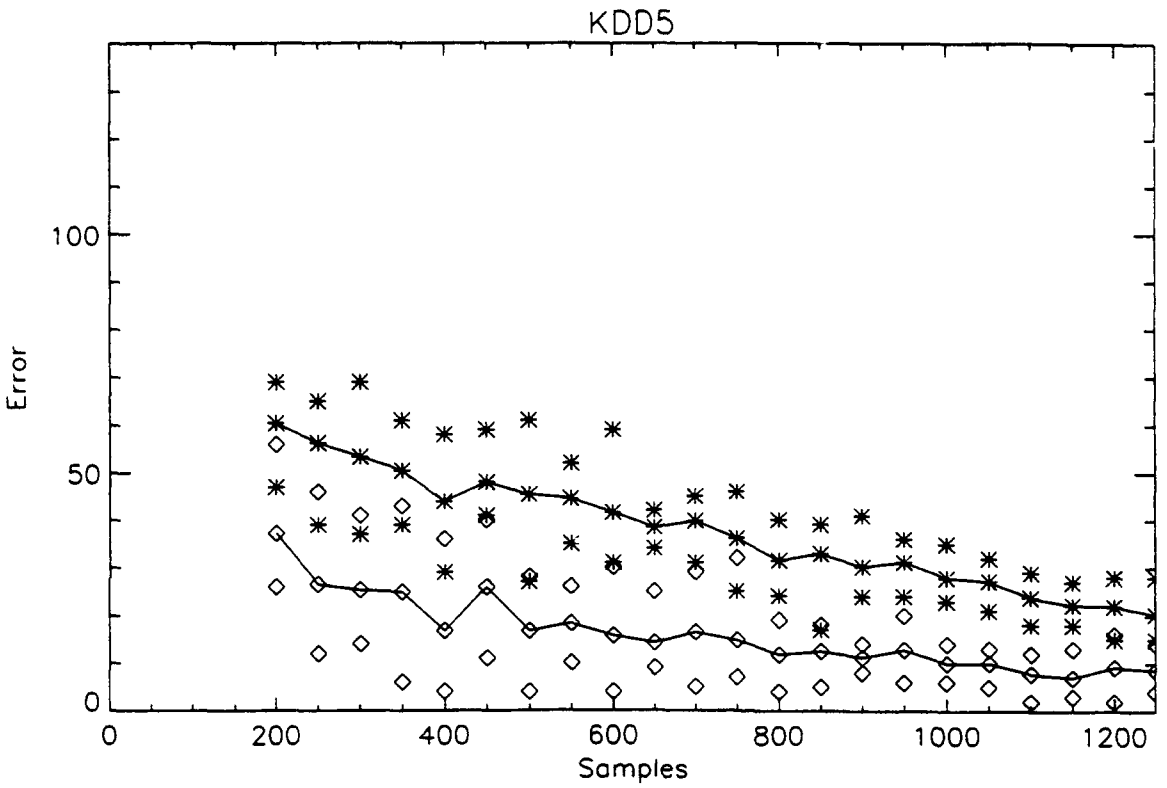
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

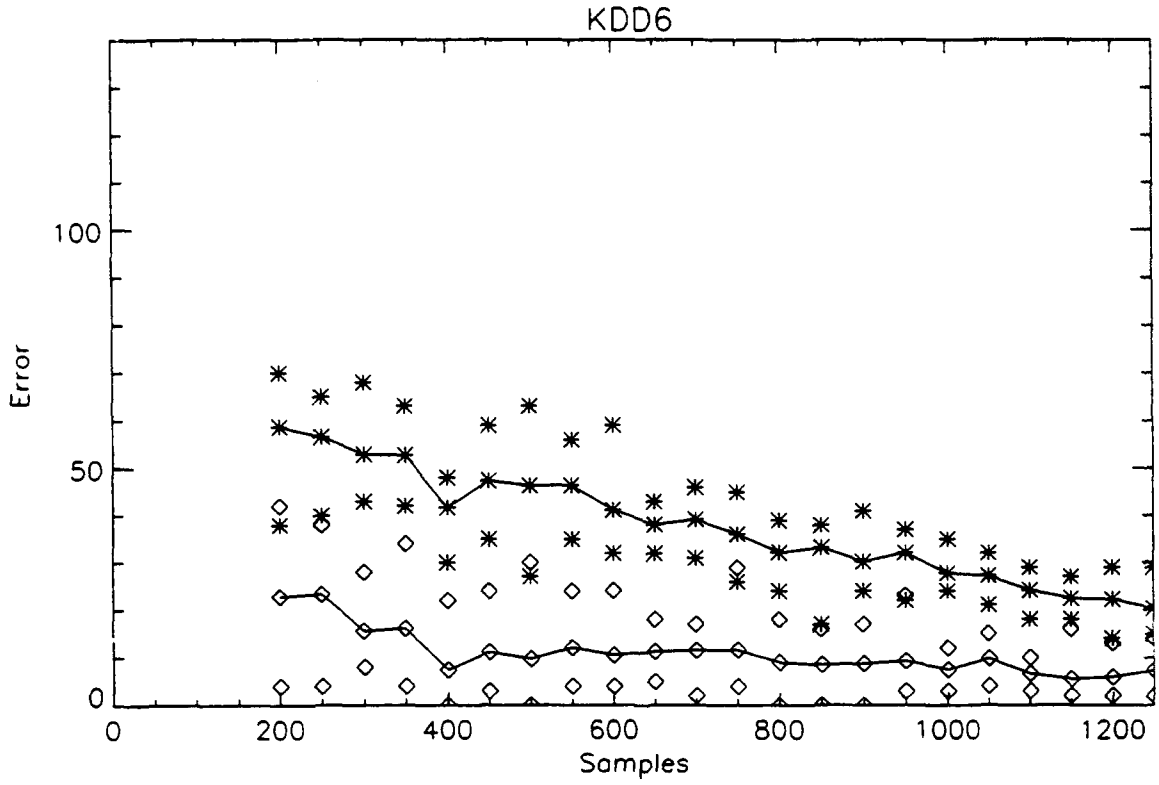
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

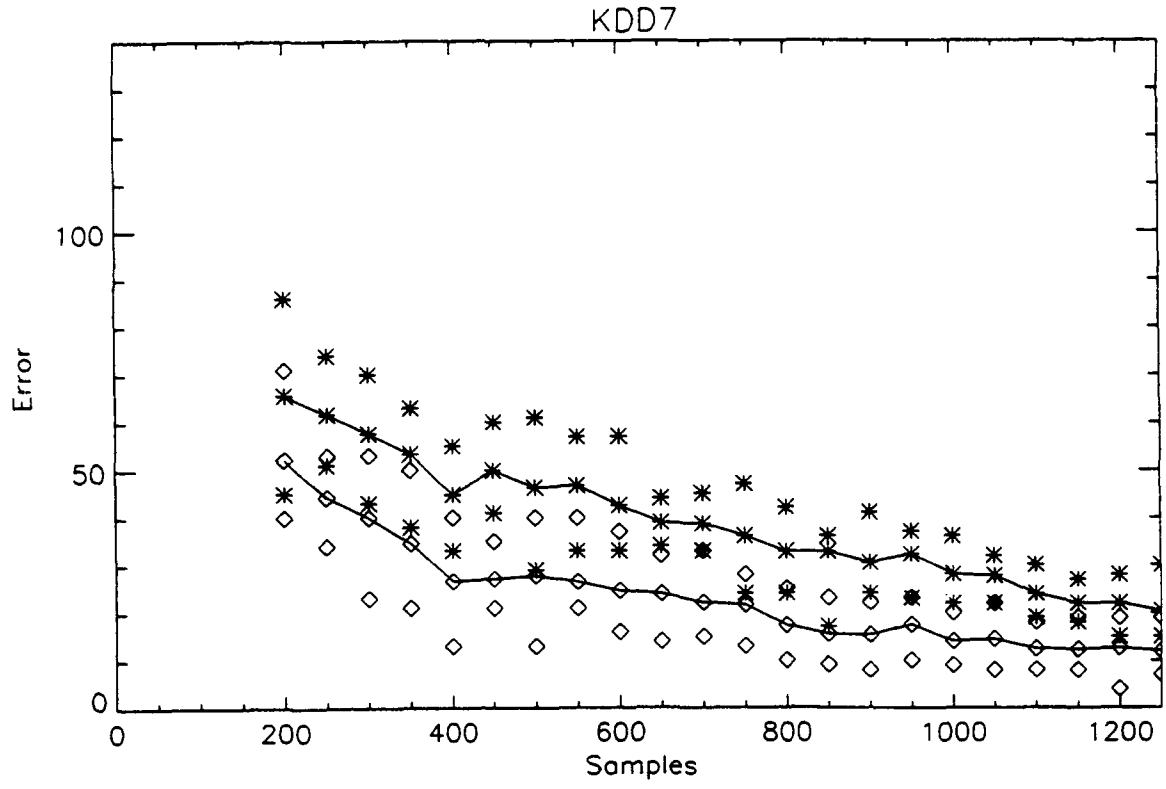
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

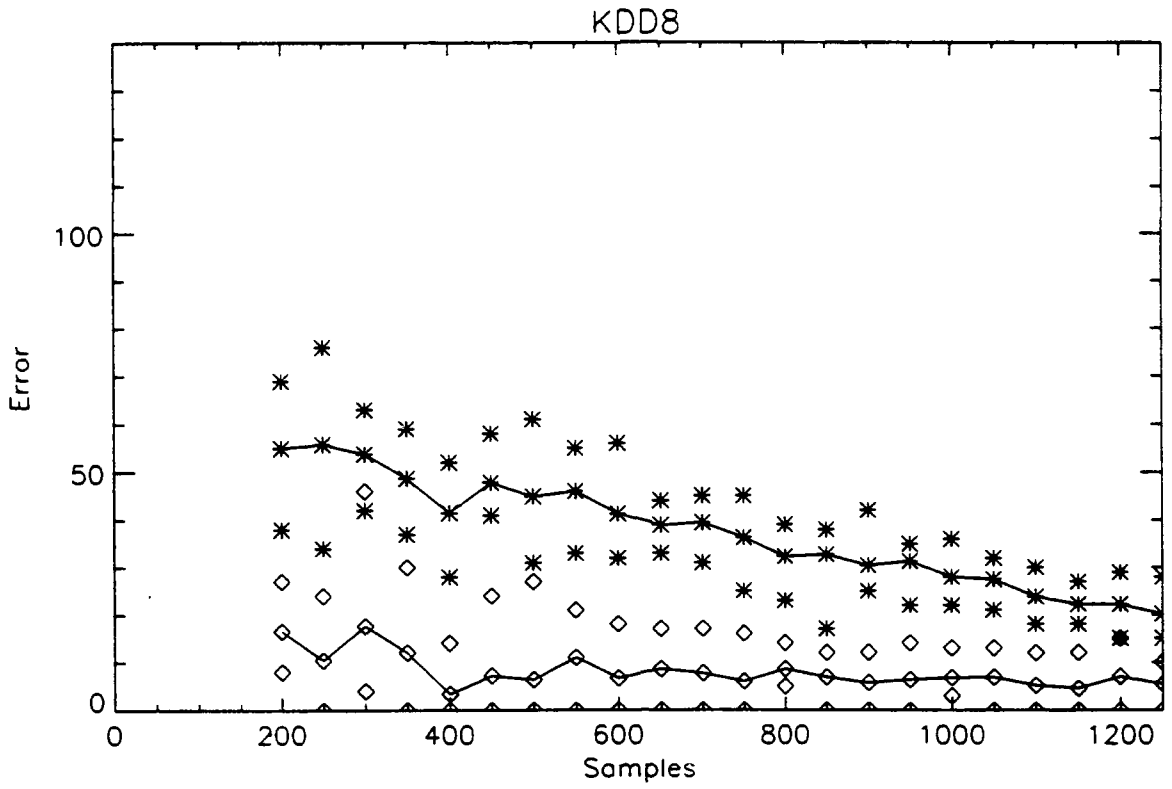
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

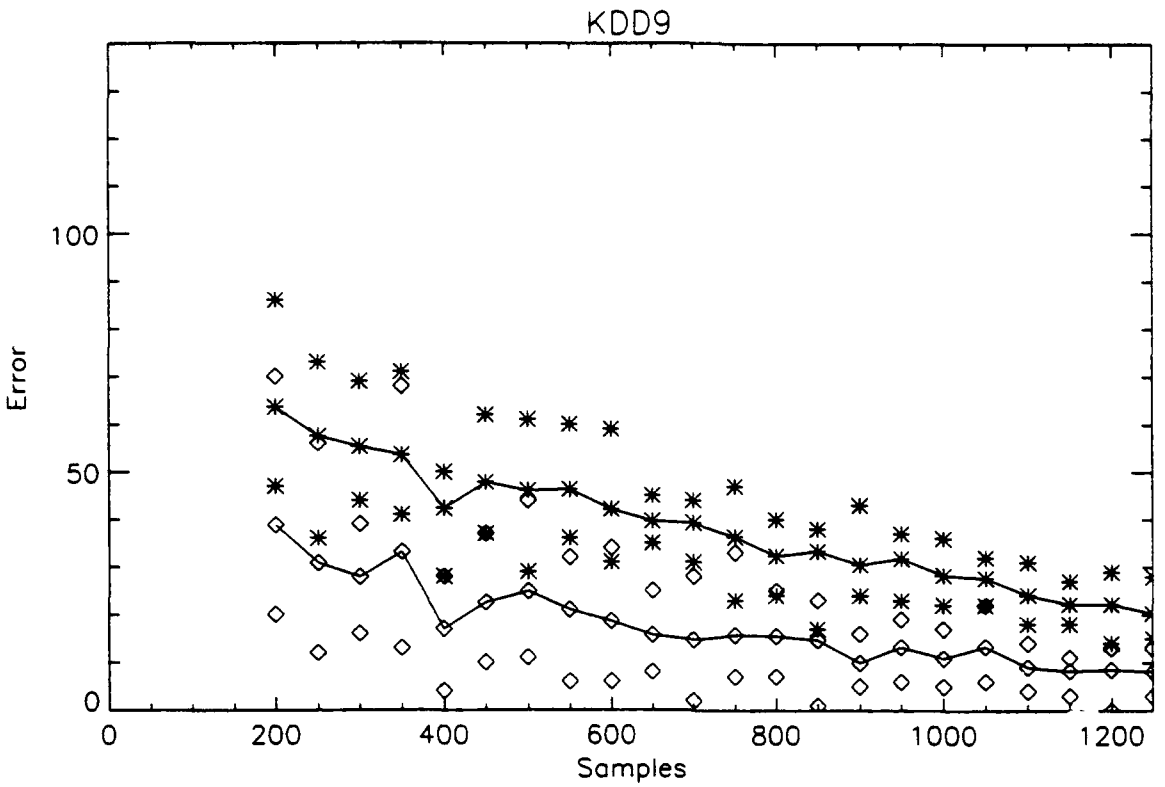
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

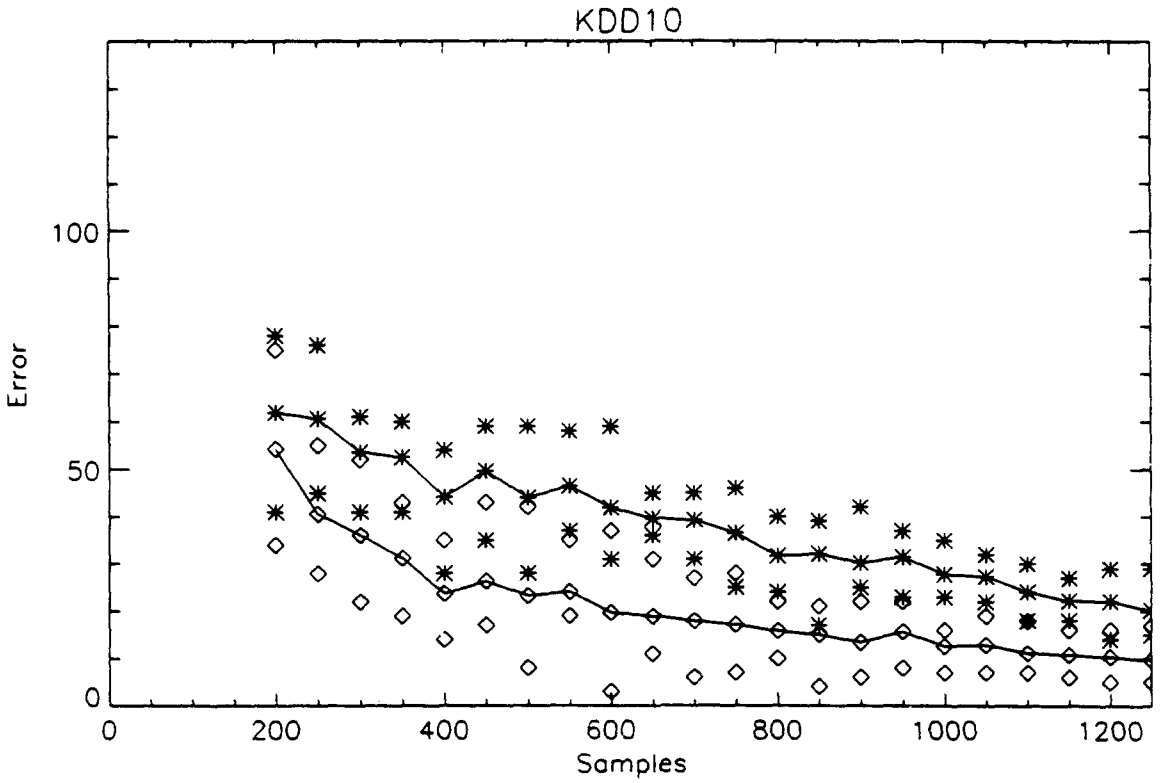
Avg error no prunning

Max error prunning

Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules



Chance

Max error no prunning

Min error no prunning

Avg error no prunning

Max error prunning

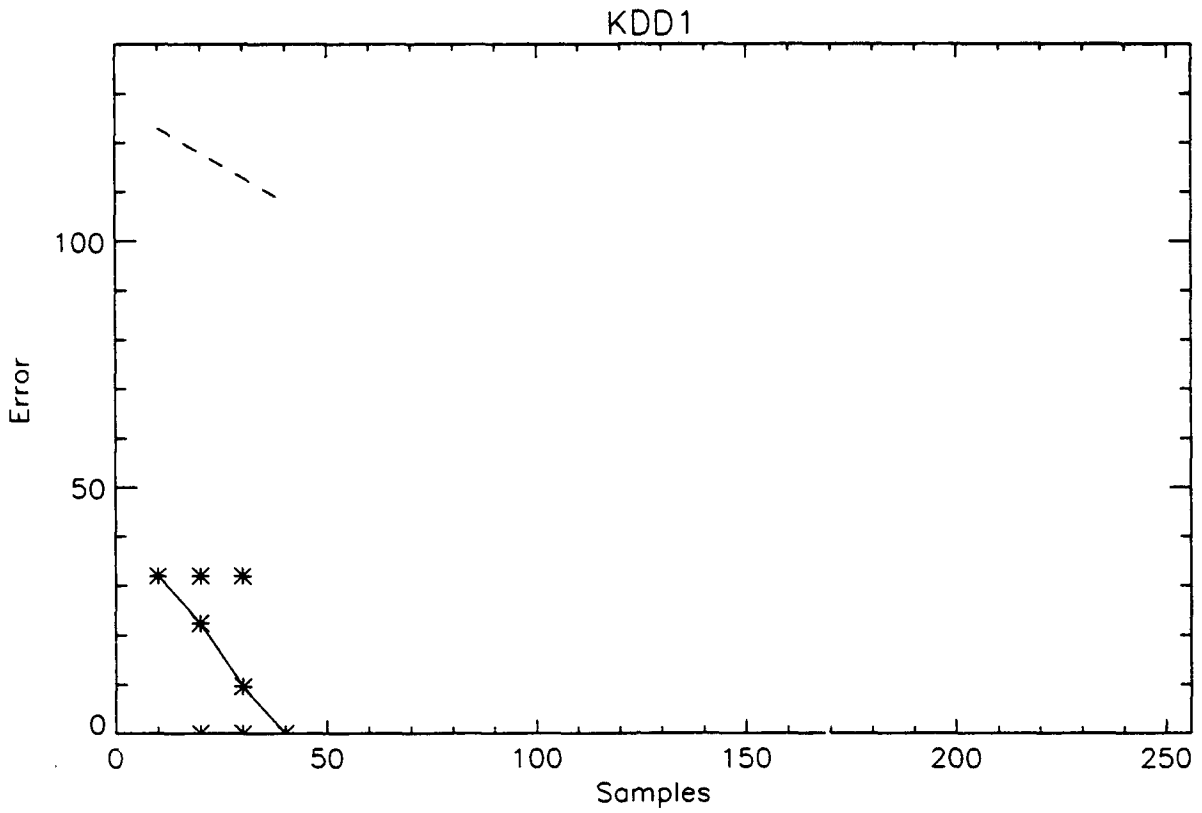
Min error prunning

Avg error prunning

C45 −t10 −m0, with replacement, 20% noise, Majority Rules

D Graphs Showing the Difference between Sampling with and without Replacement

In Appendix D, we show learning curves without any replacement and without any noise. The second set shows learning with replacement but without noise.



Chance

Max error

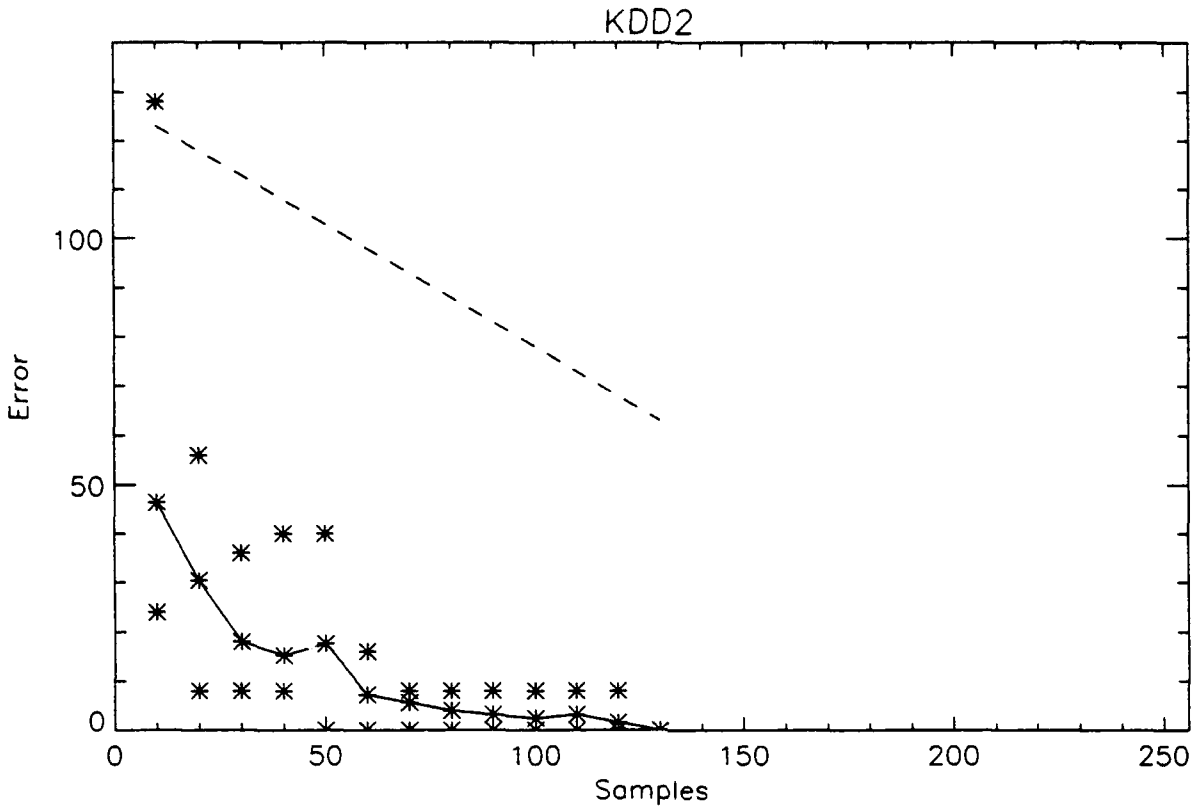
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

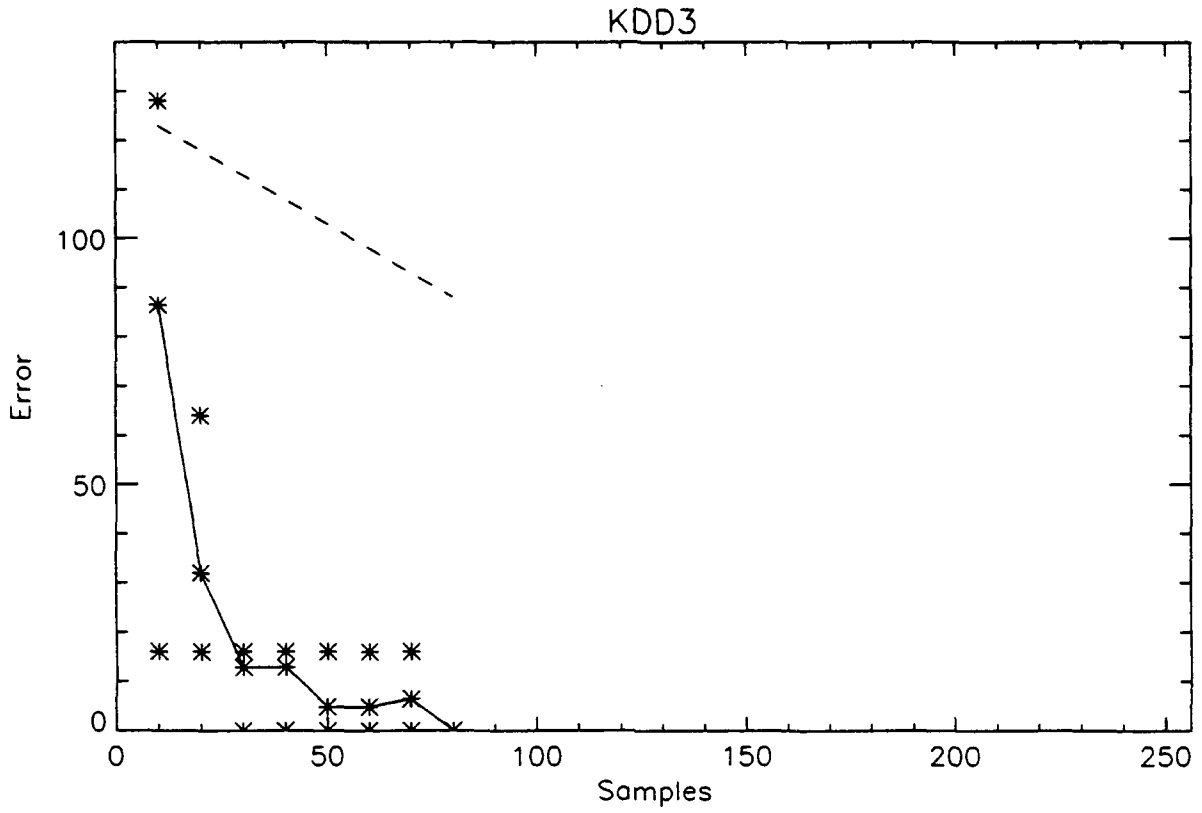
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

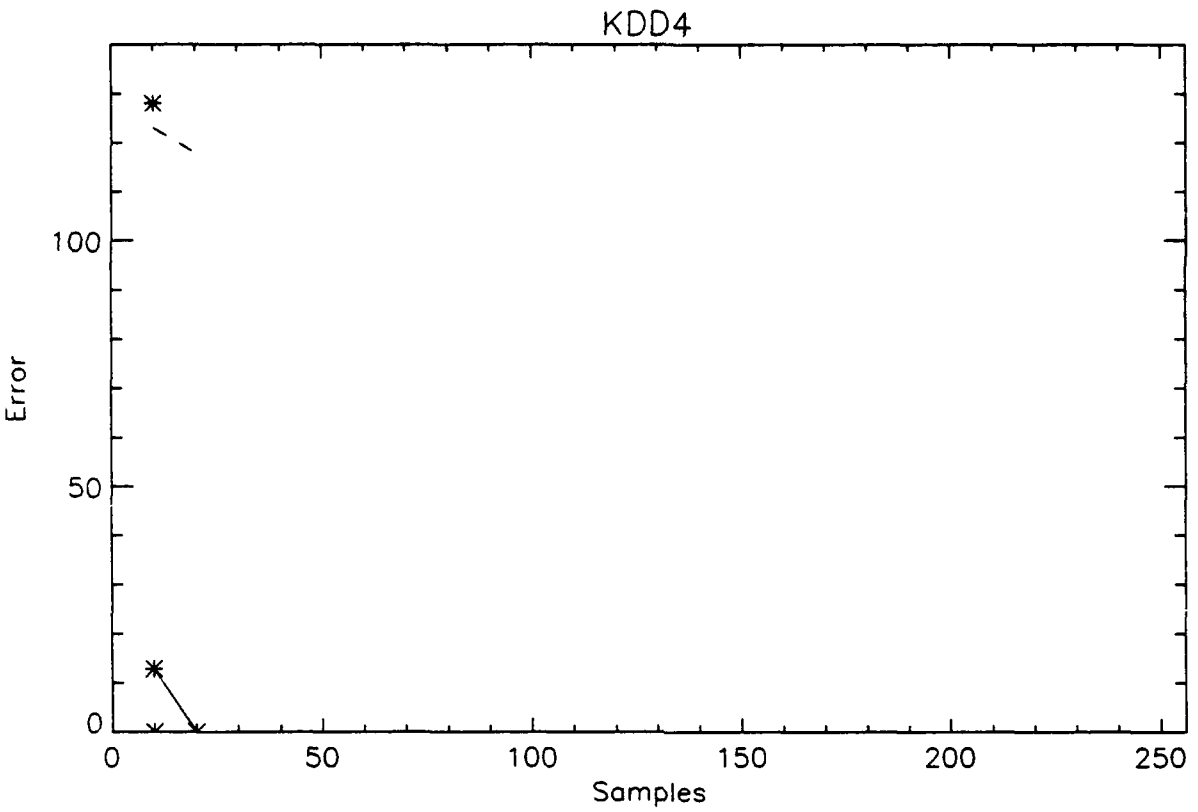
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

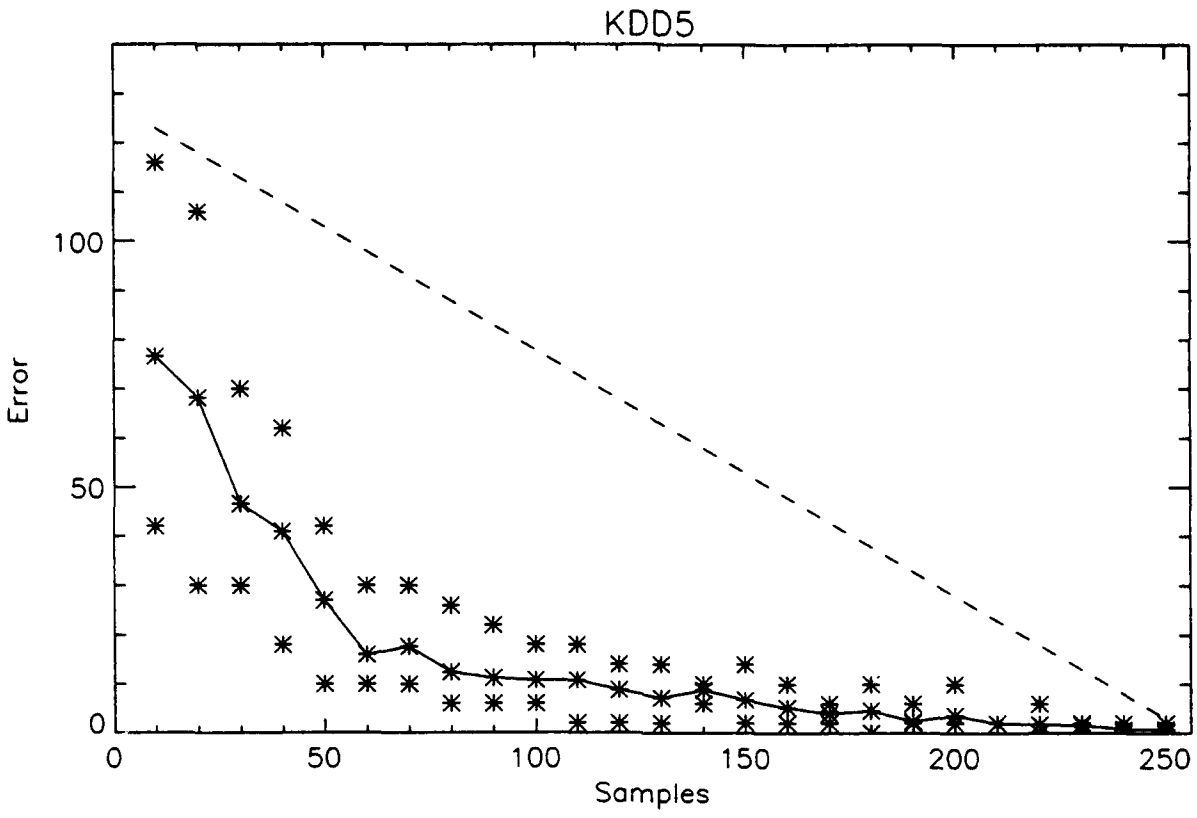
Min error

Avg error

C45 -t10

no replacement

nc noise



Chance

Max error

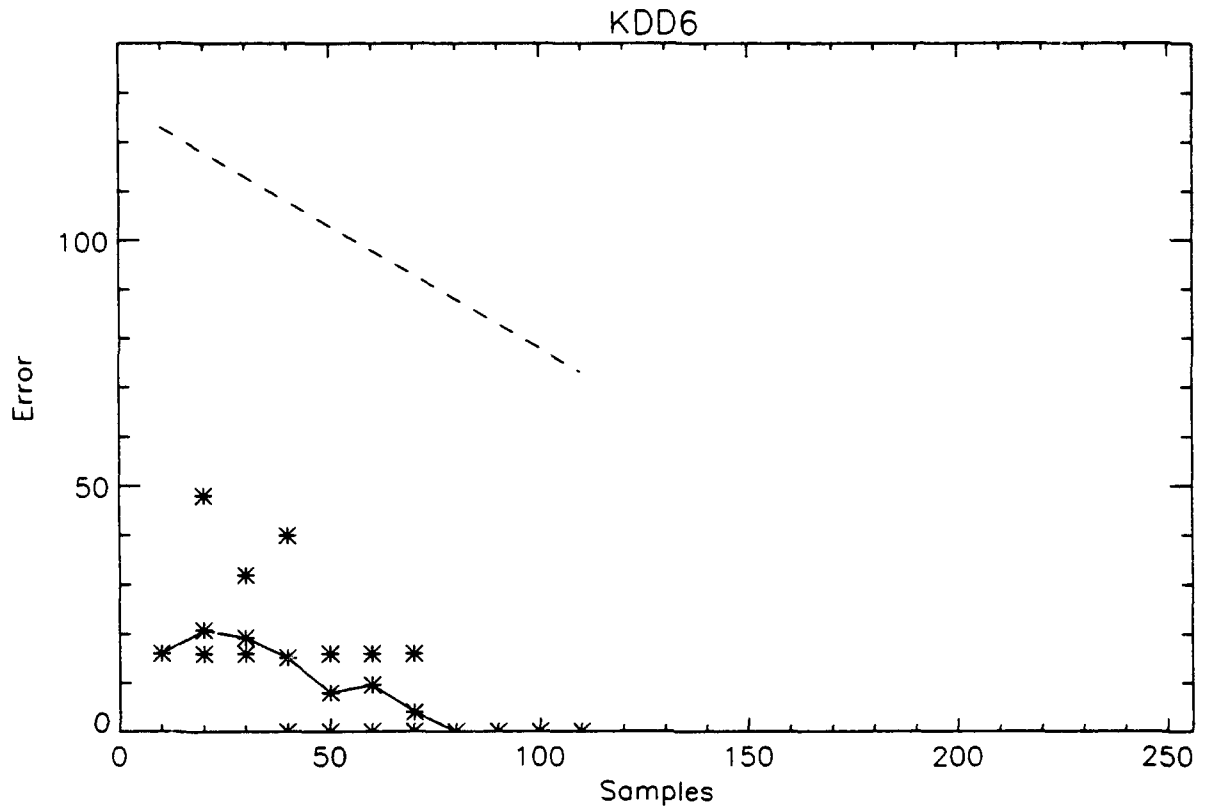
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

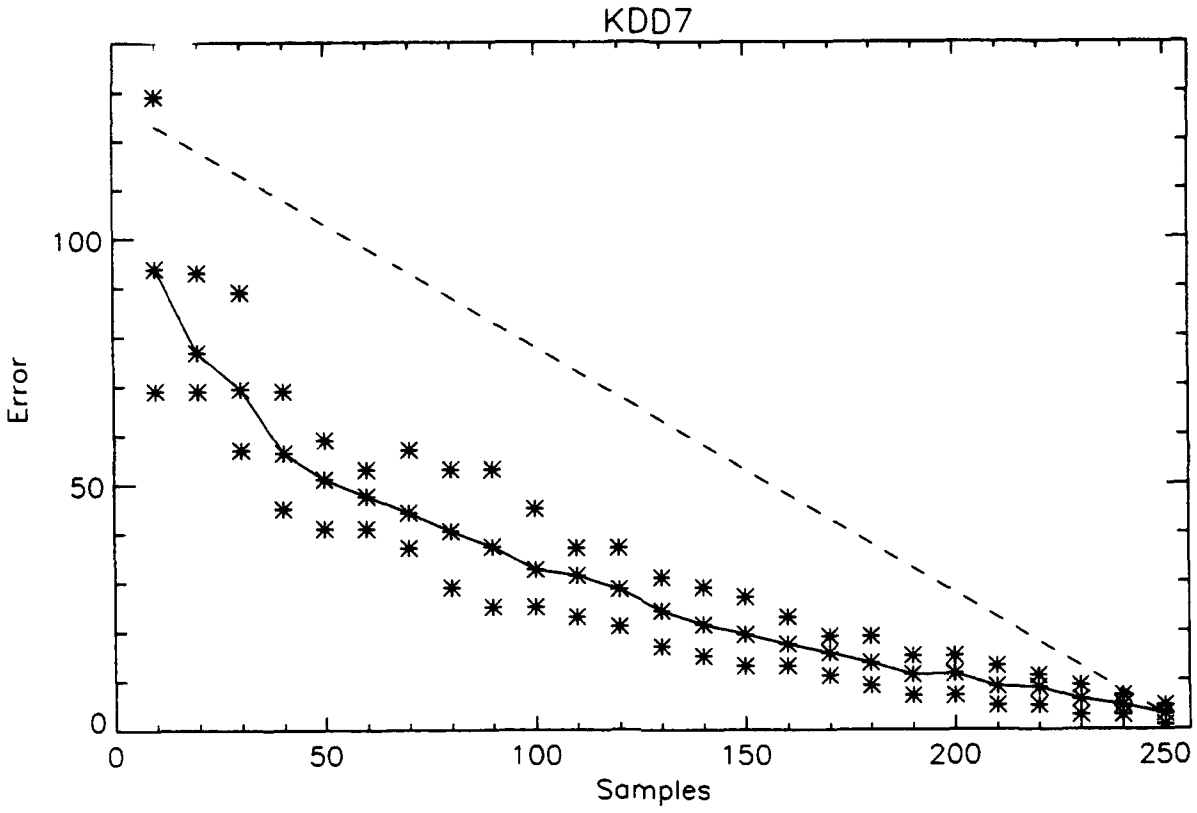
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

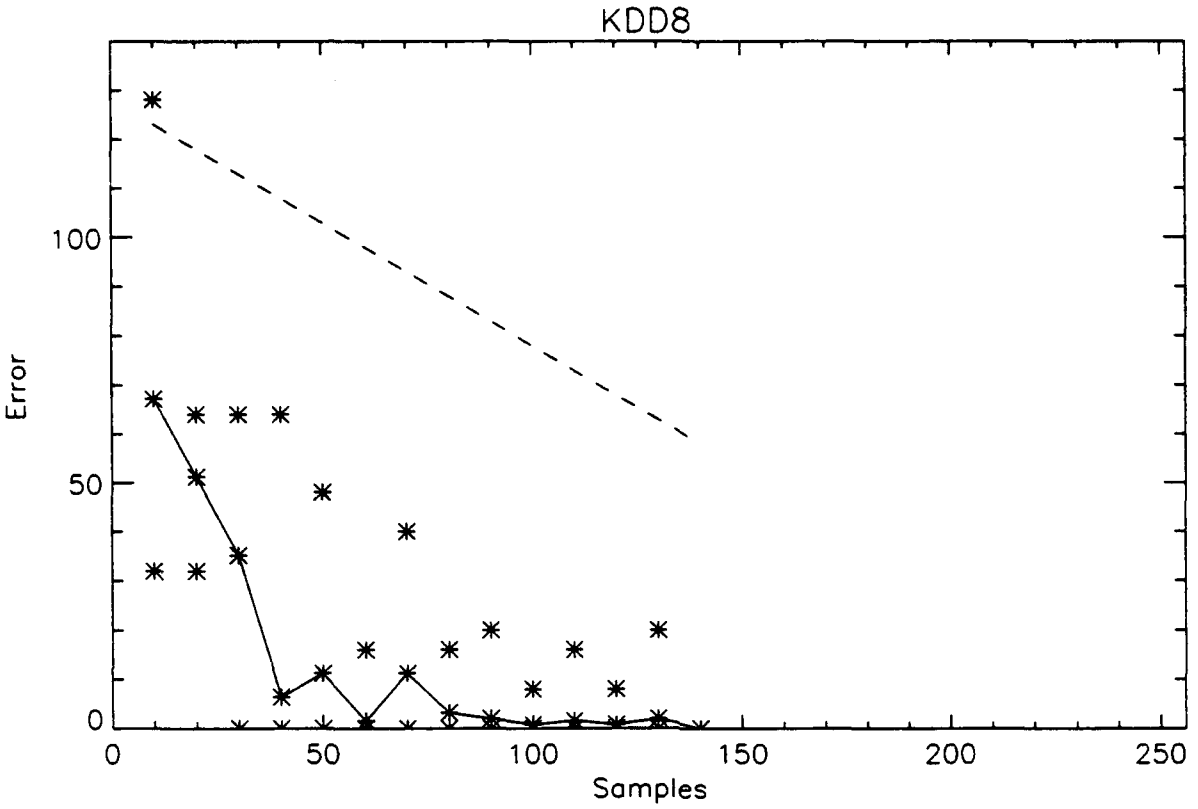
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

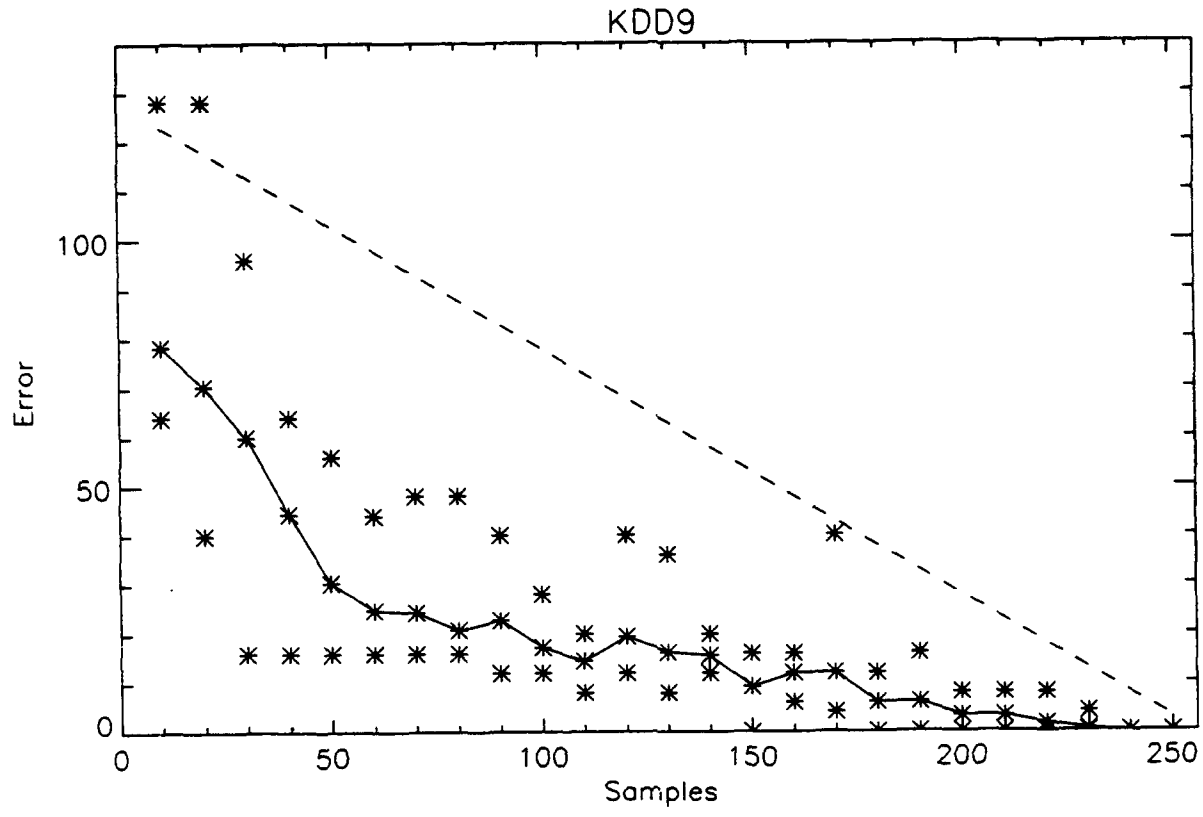
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

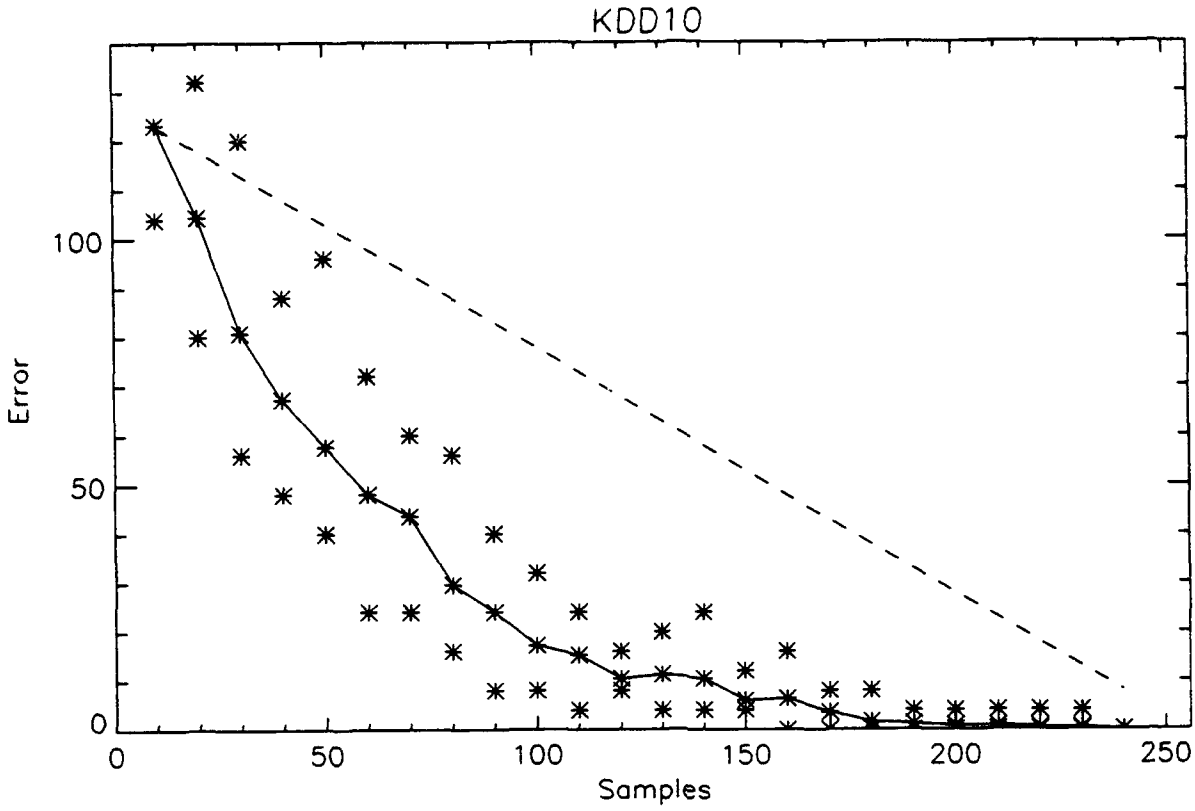
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

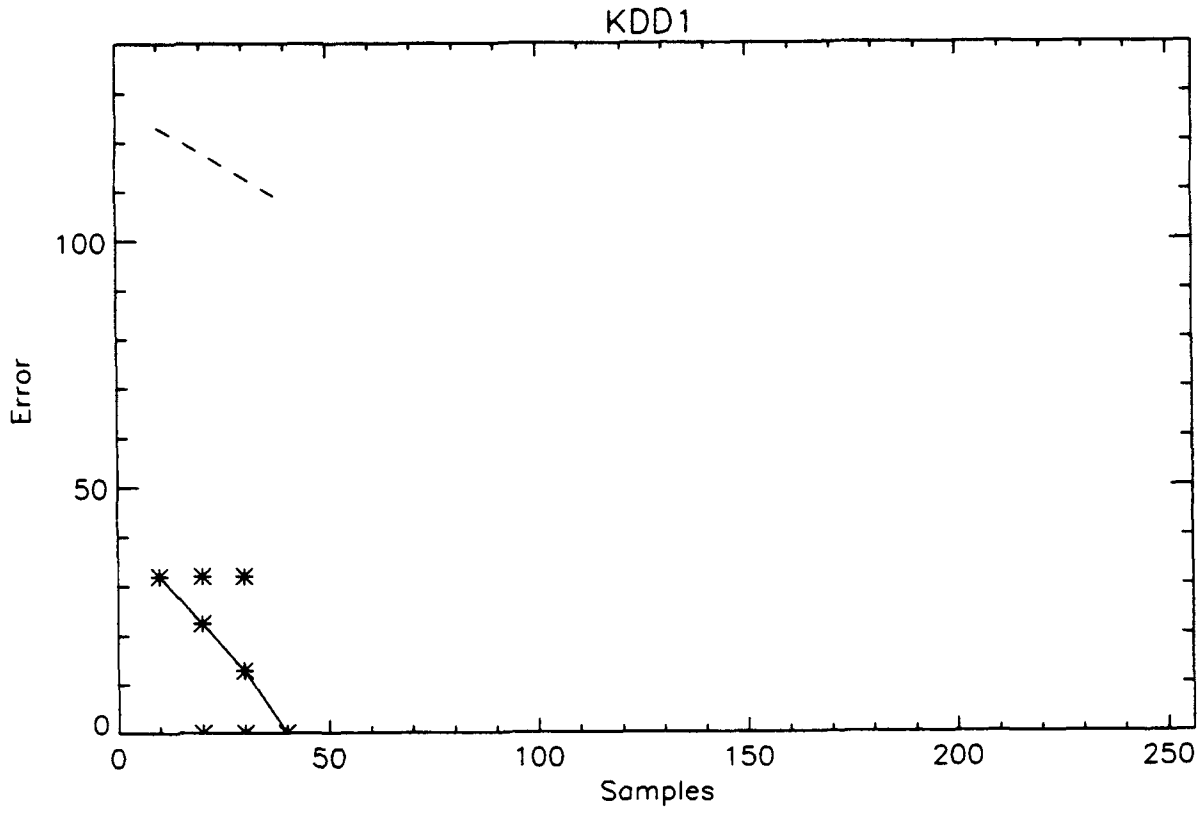
Min error

Avg error

C45 -t10

no replacement

no noise



Chance

Max error

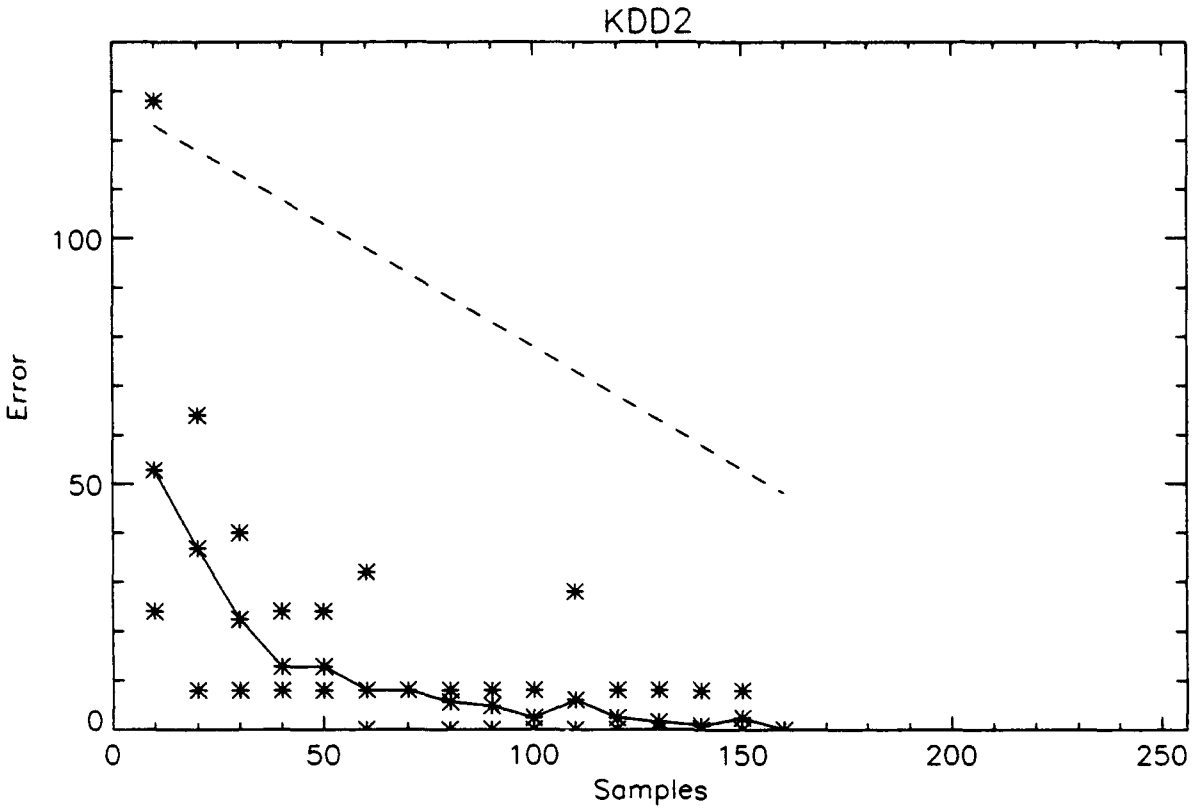
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

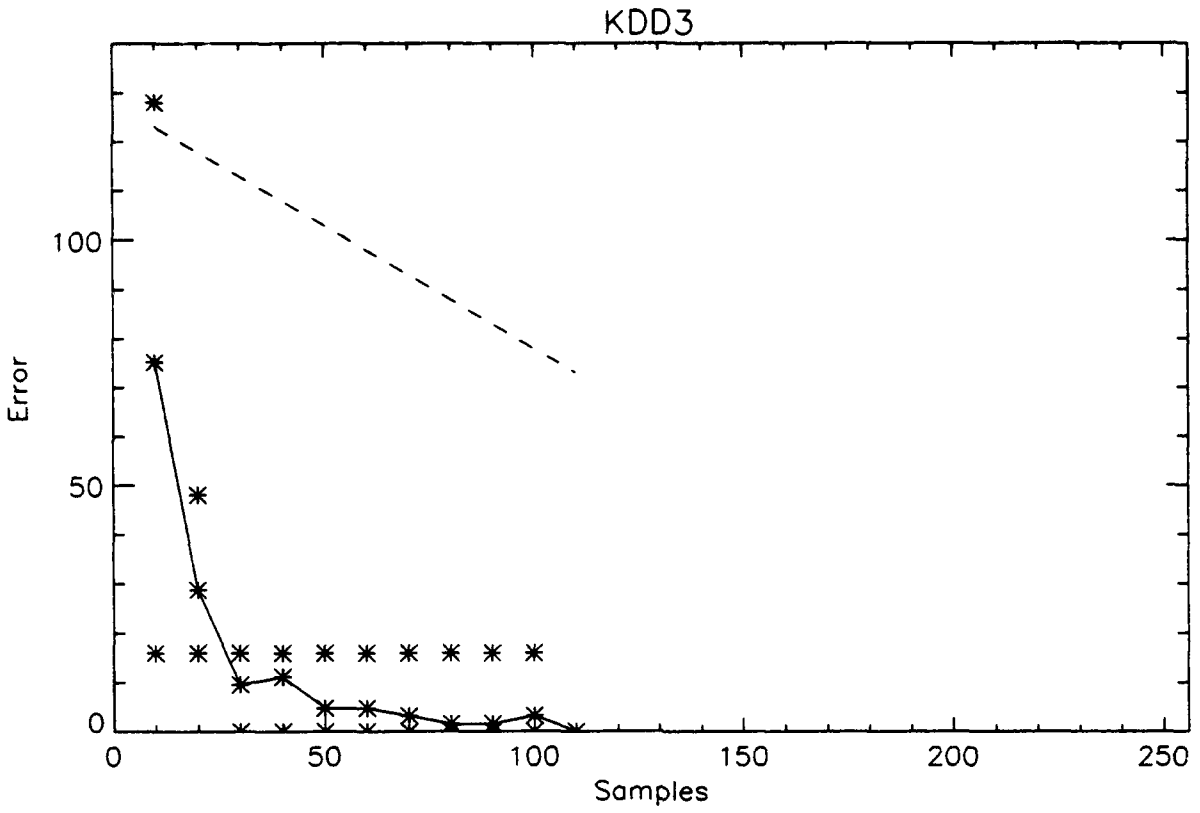
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

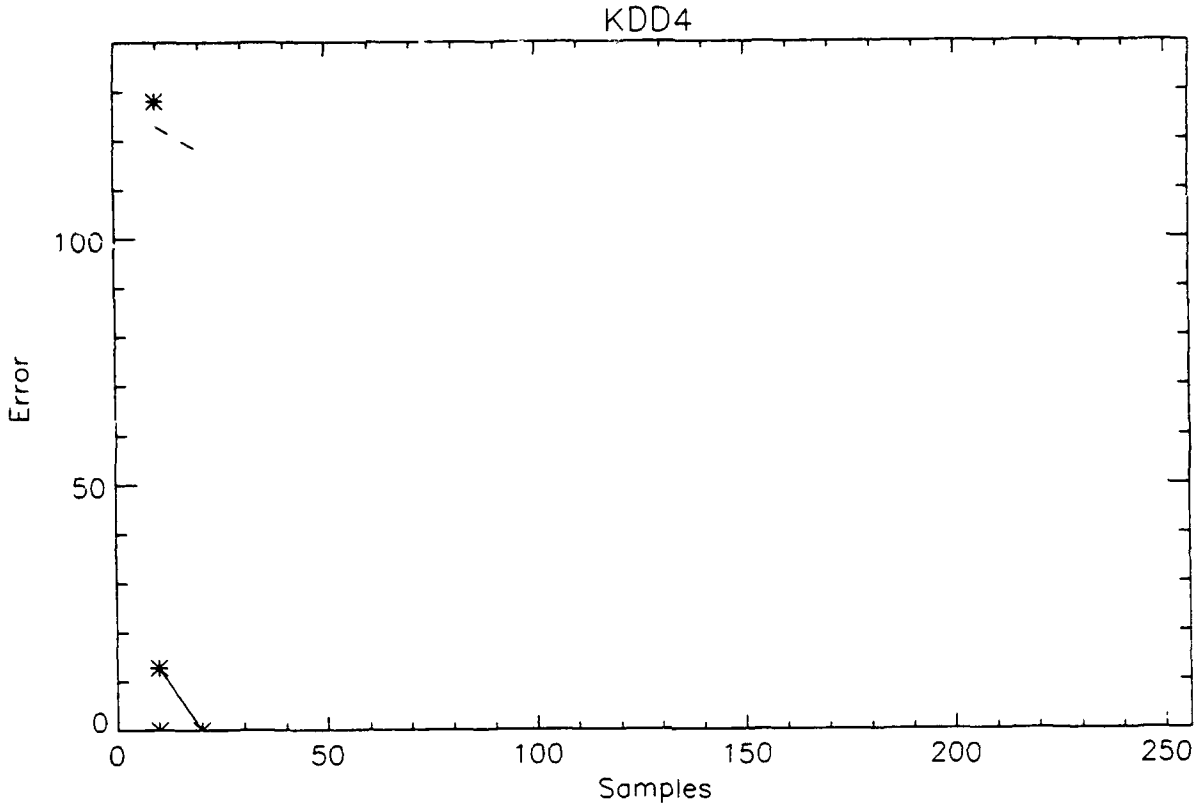
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

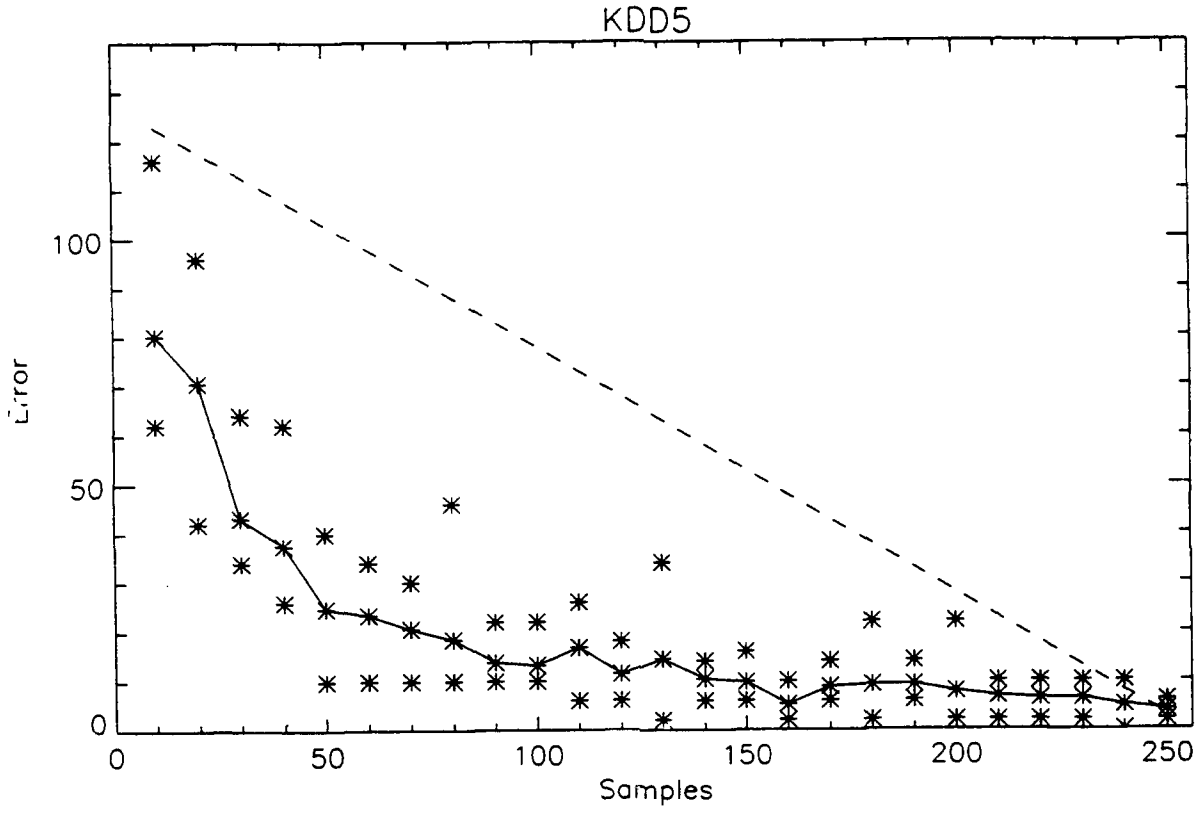
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

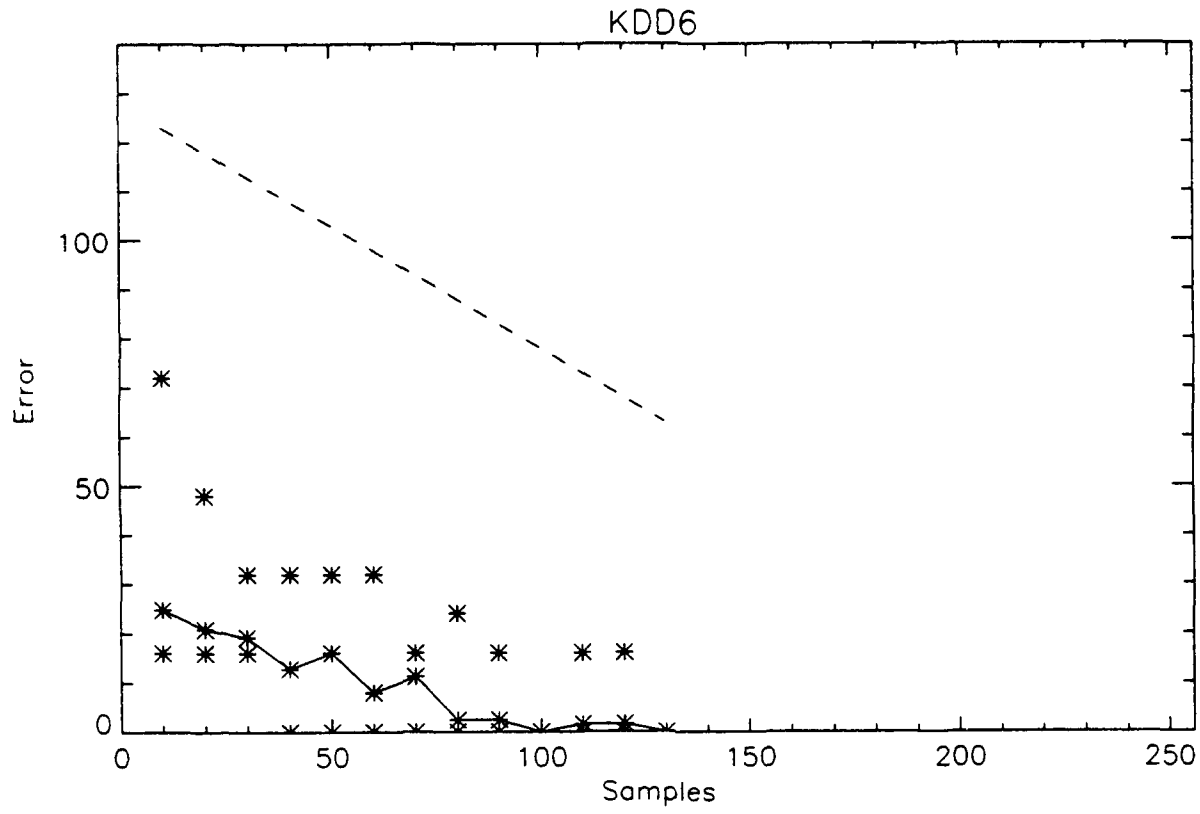
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

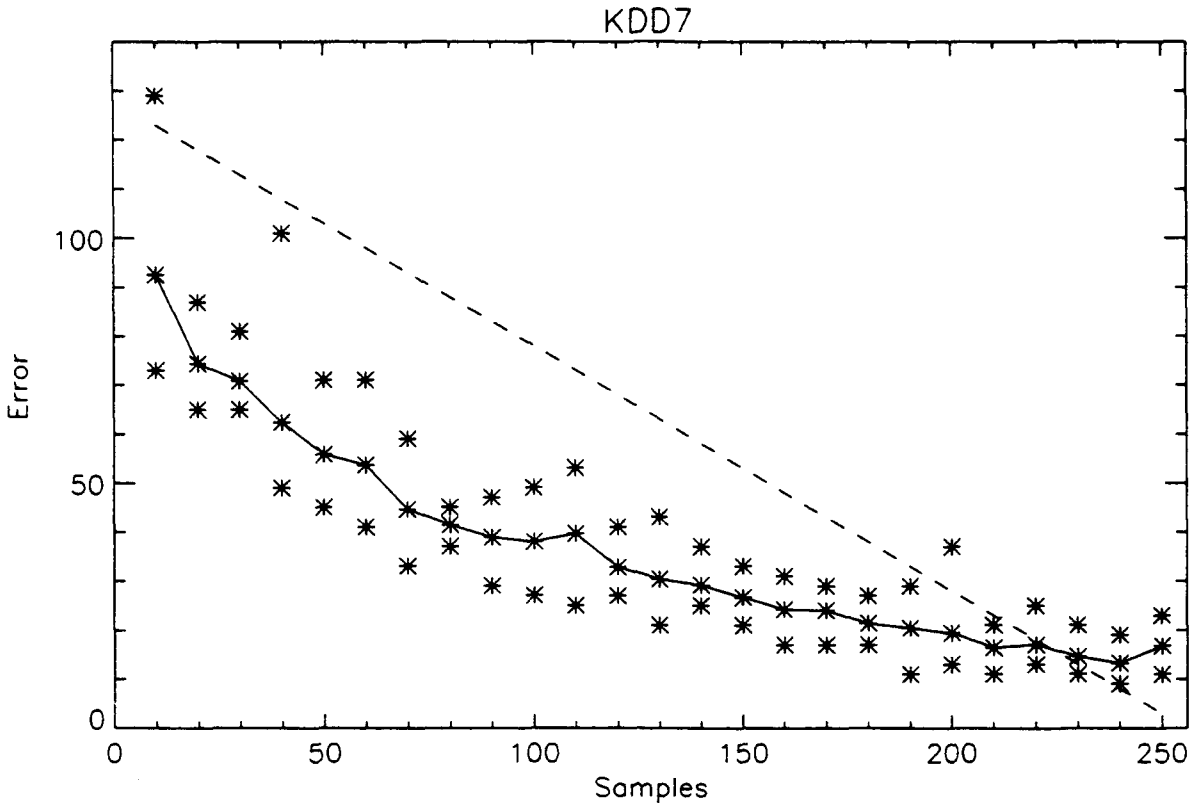
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

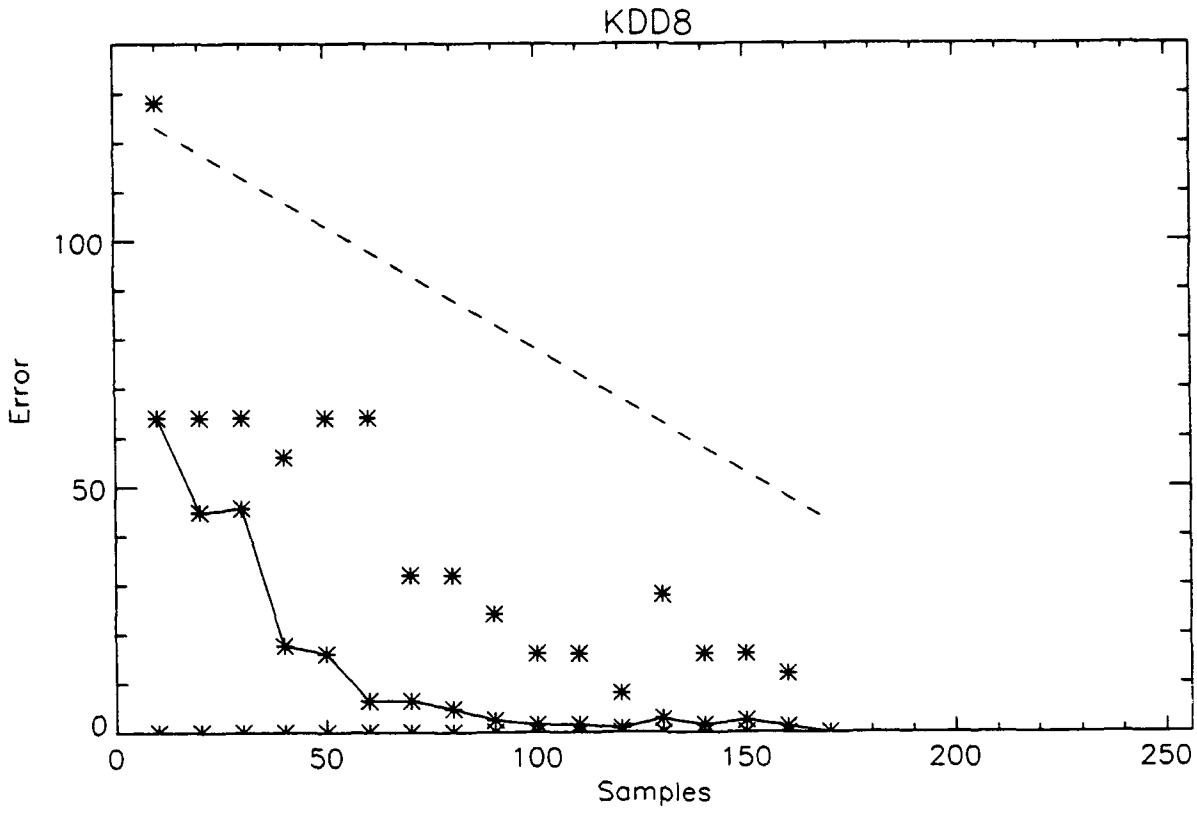
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

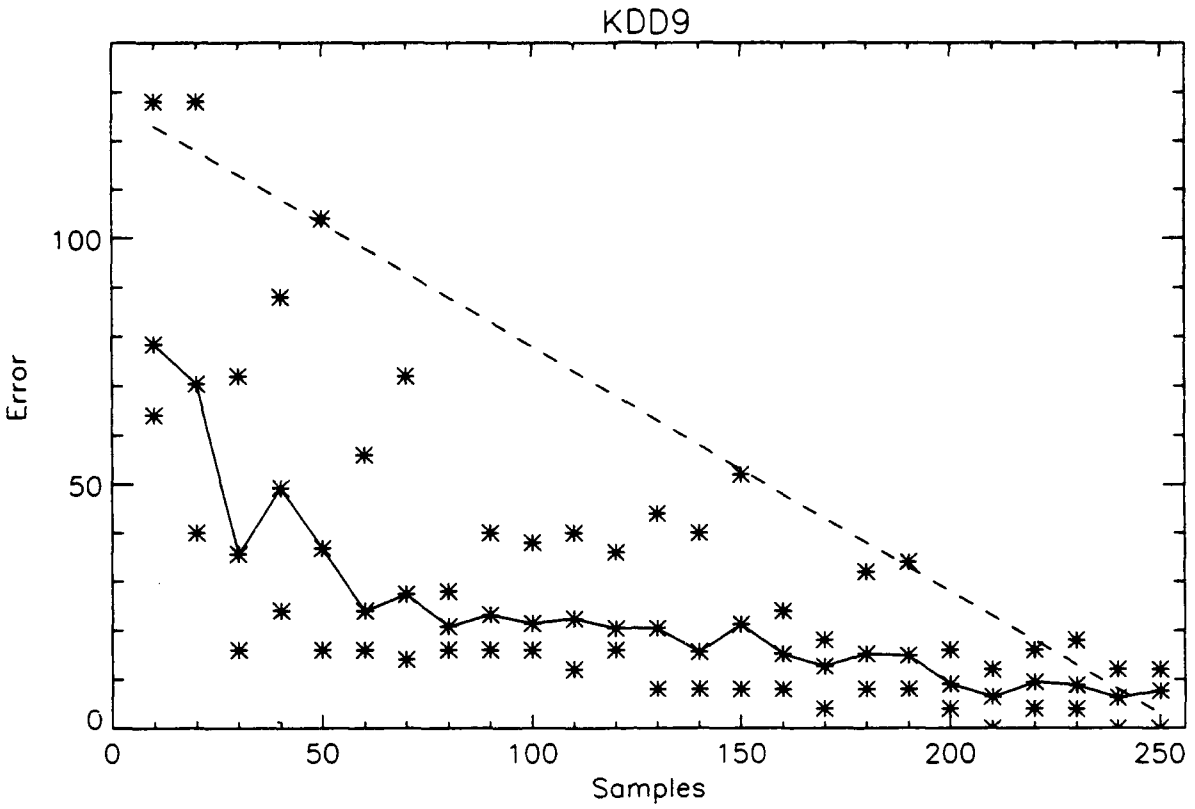
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

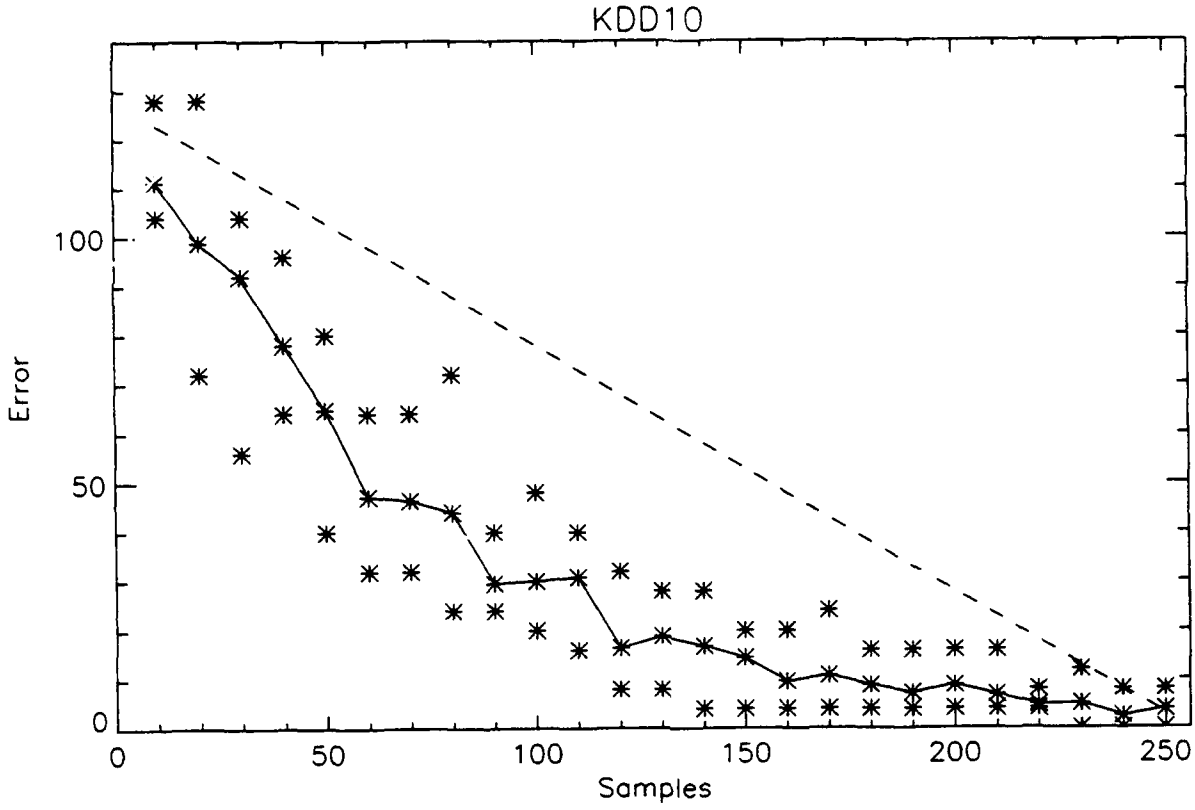
Min error

Avg error

C45 -t10

with replacement

no noise



Chance

Max error

Min error

Avg error

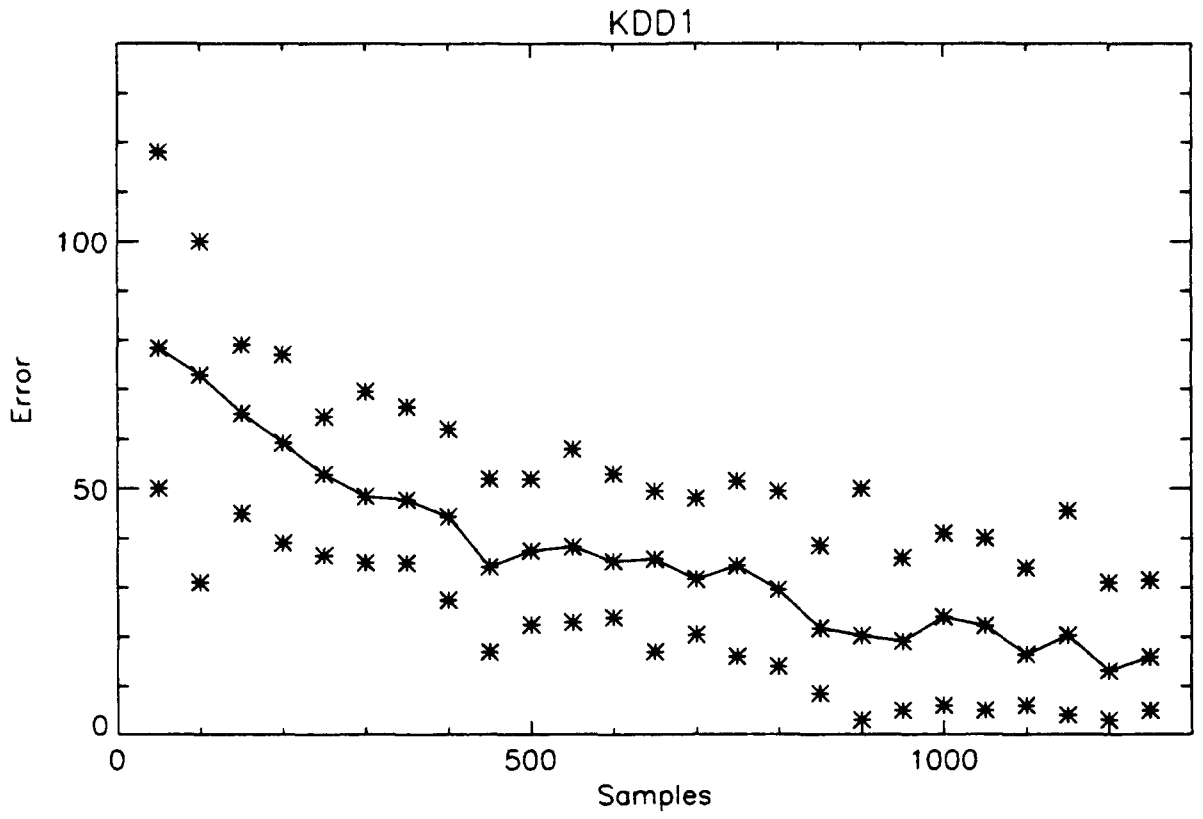
C45 -t10

with replacement

no noise

E Noise Experiments with FLASH

Finally, in Appendix E, we show some learning curves for FLASH (function decomposition), with noise. The extensions were made so as to allow FLASH to learn with noise. Specifically, in the first set, we removed all conflicting data elements and then we removed all duplicate data. In the second set, we choose the majority occurring element and then removed all duplicate data.



Max error

Min error

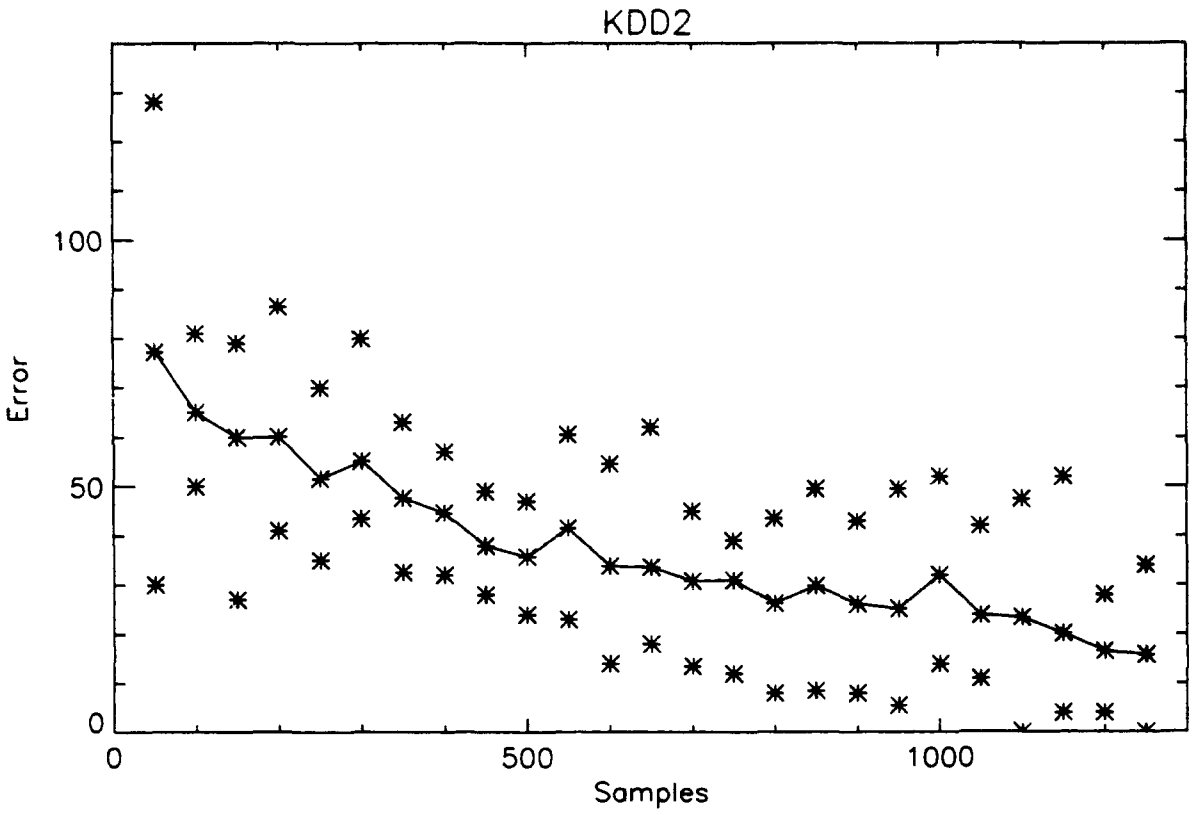
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

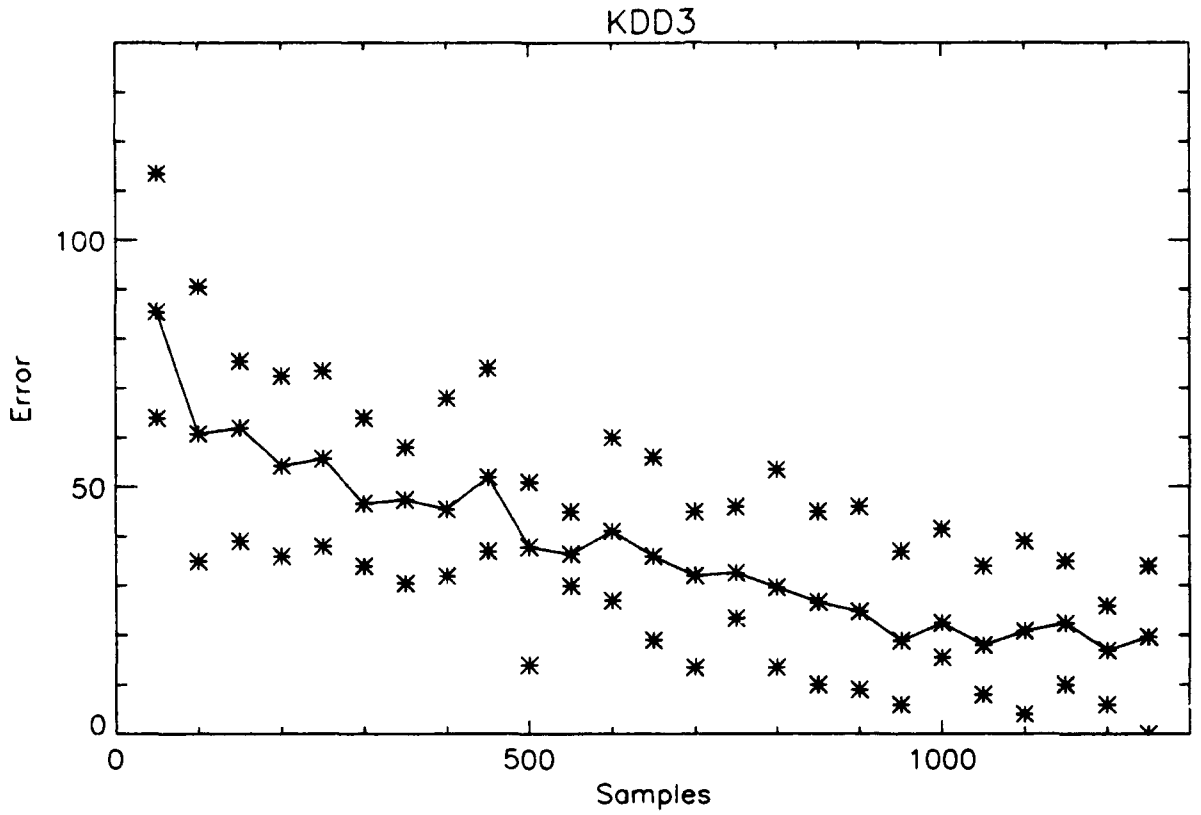
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

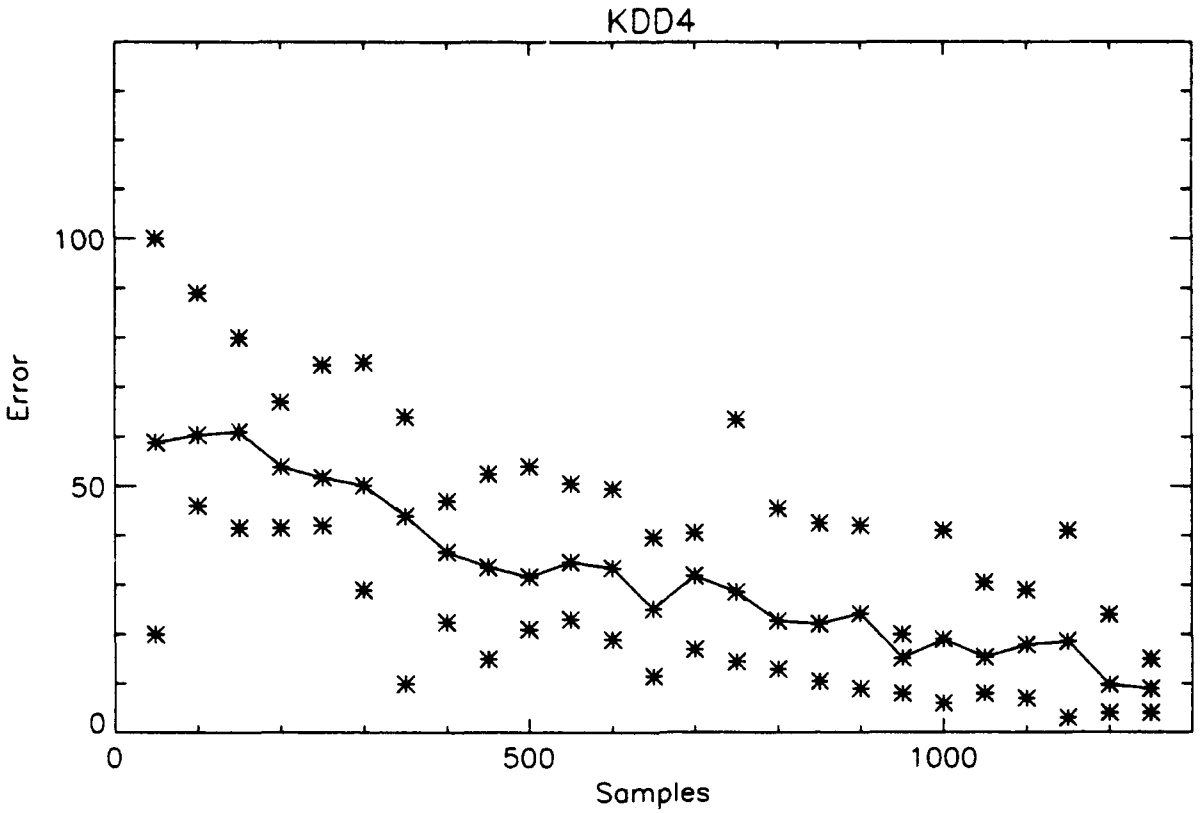
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

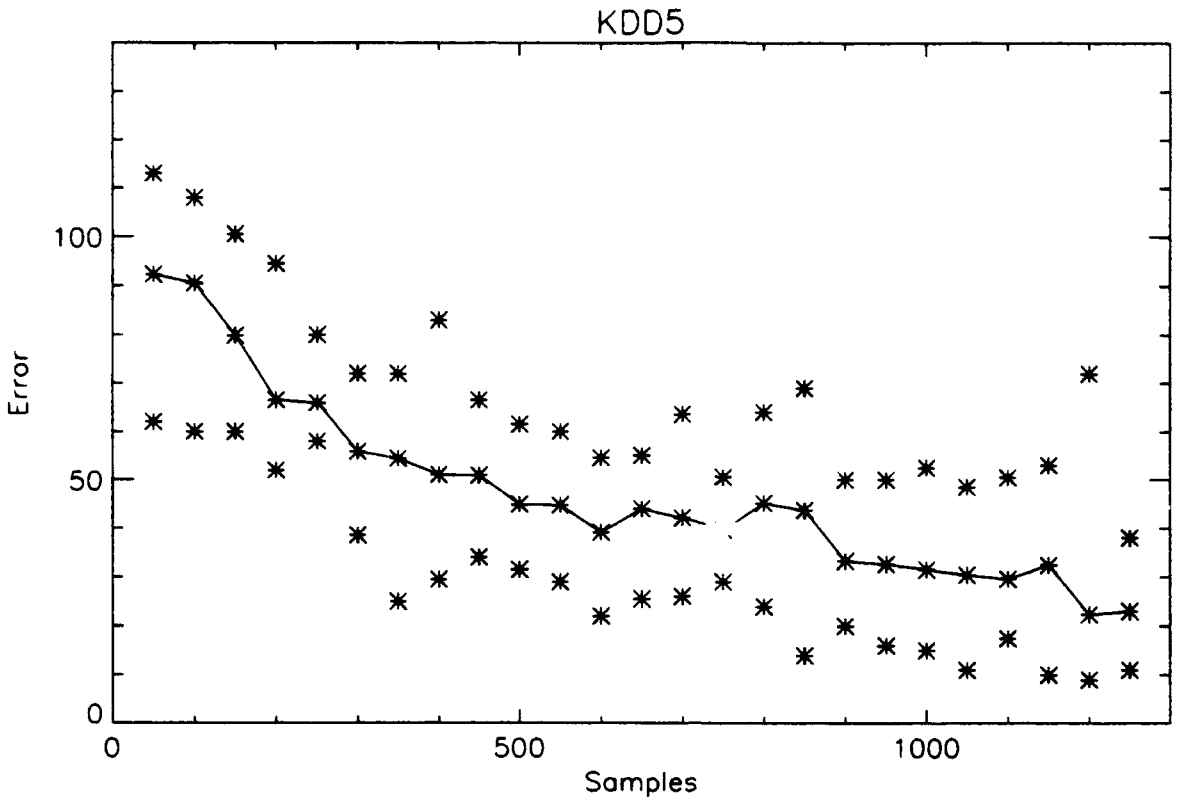
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

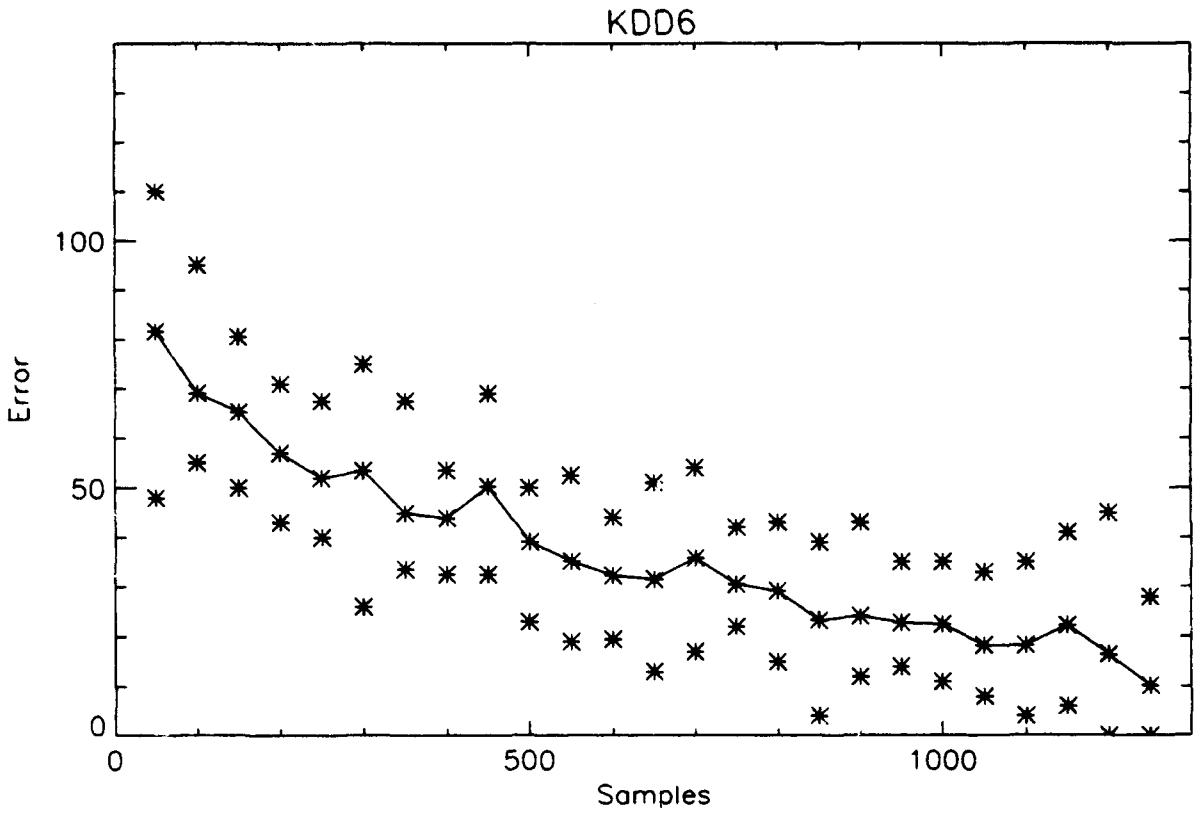
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

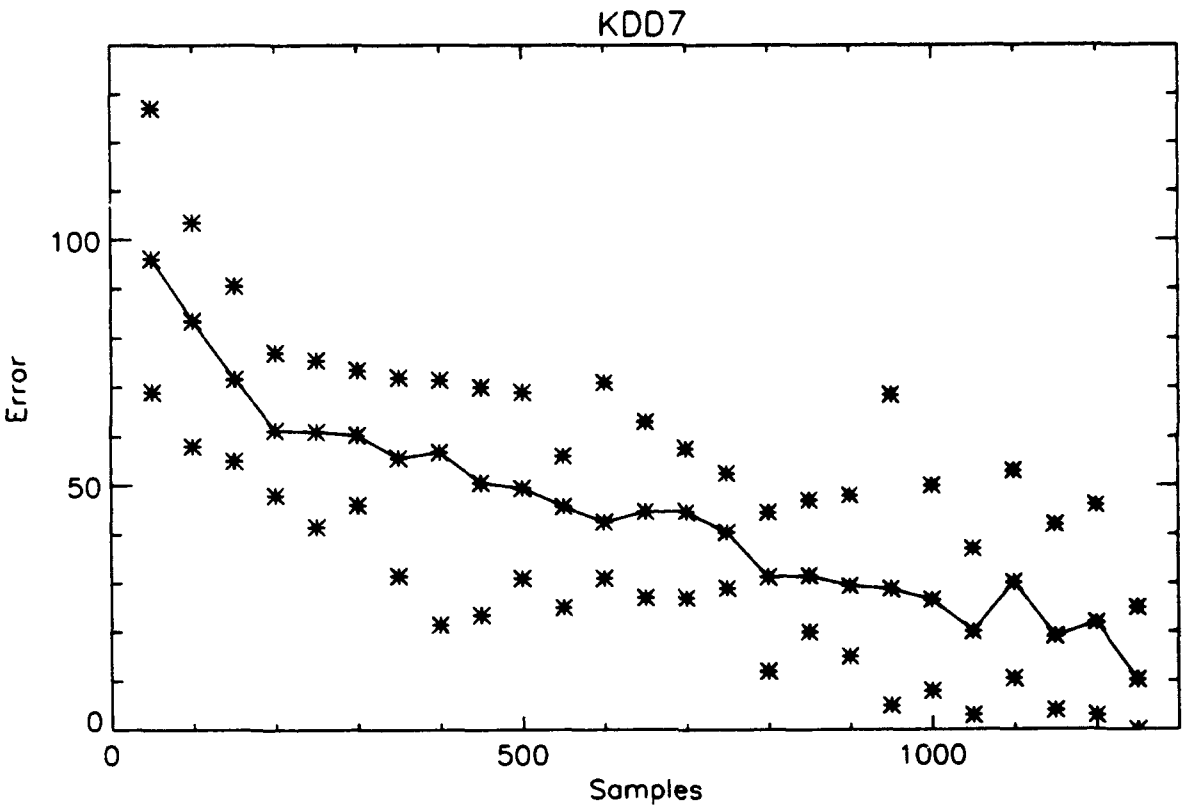
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

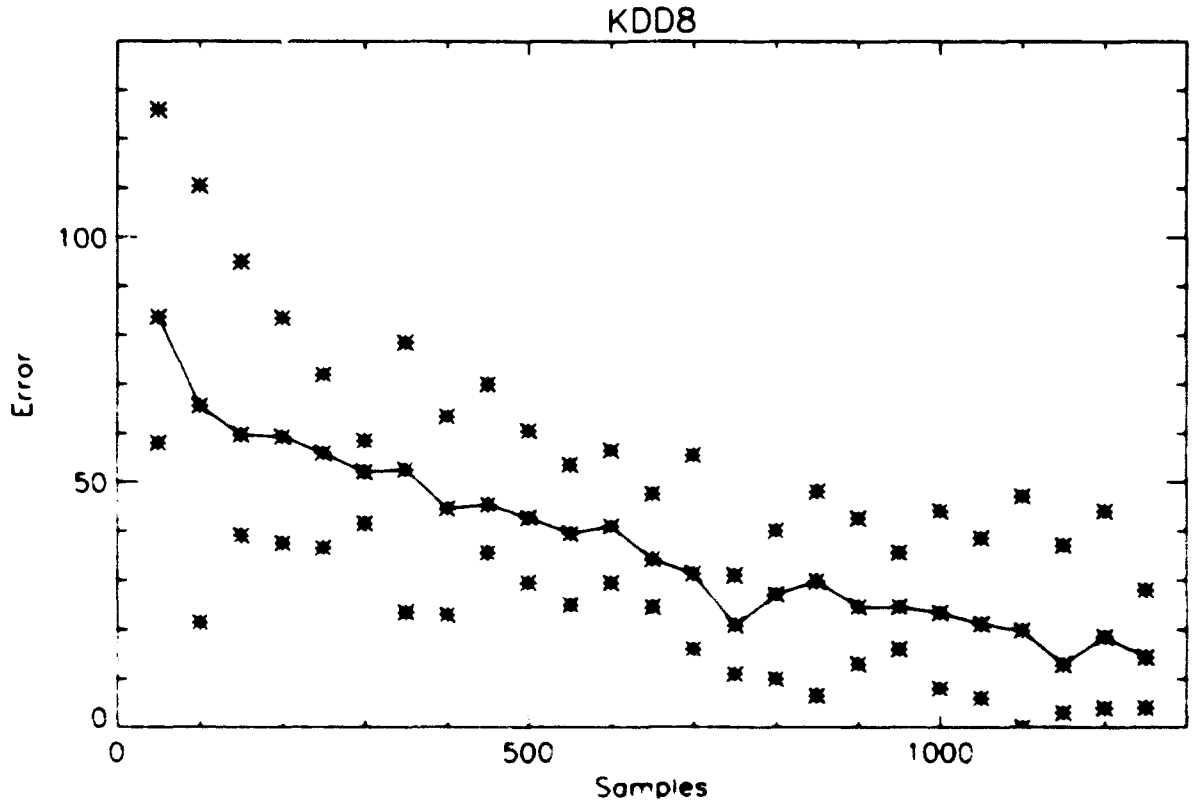
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

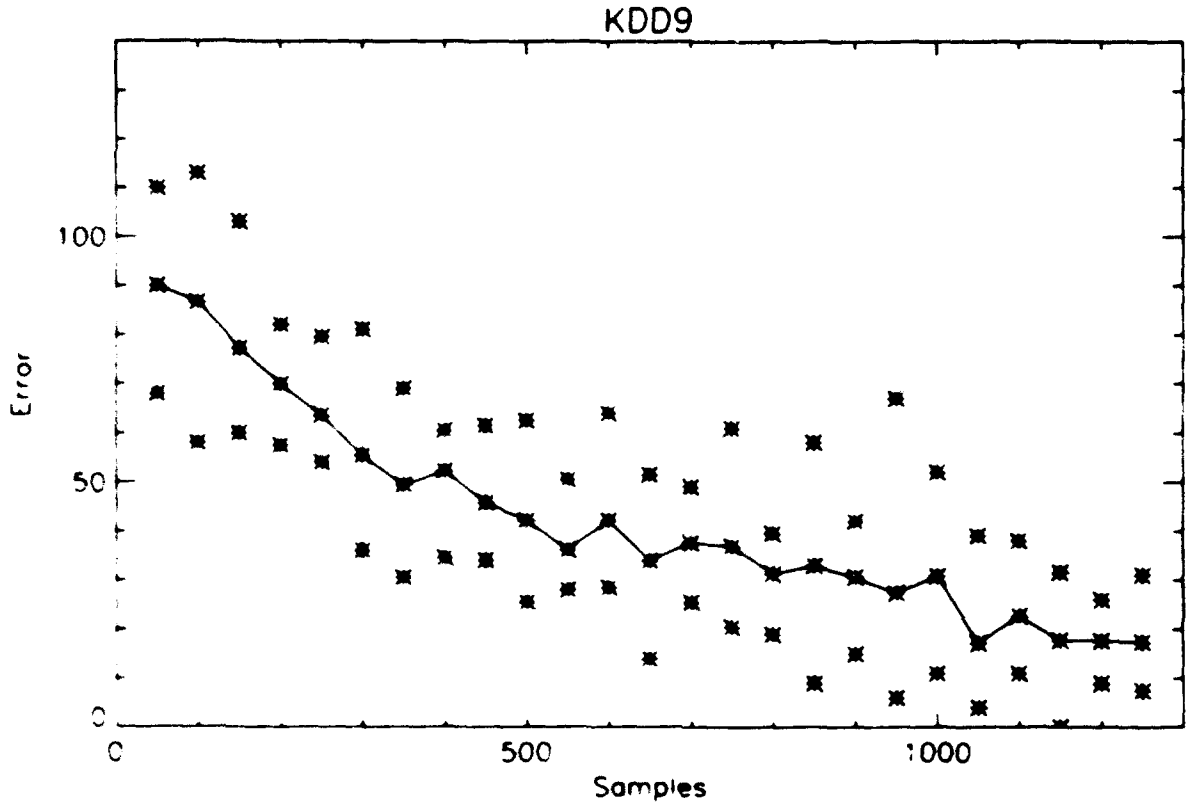
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

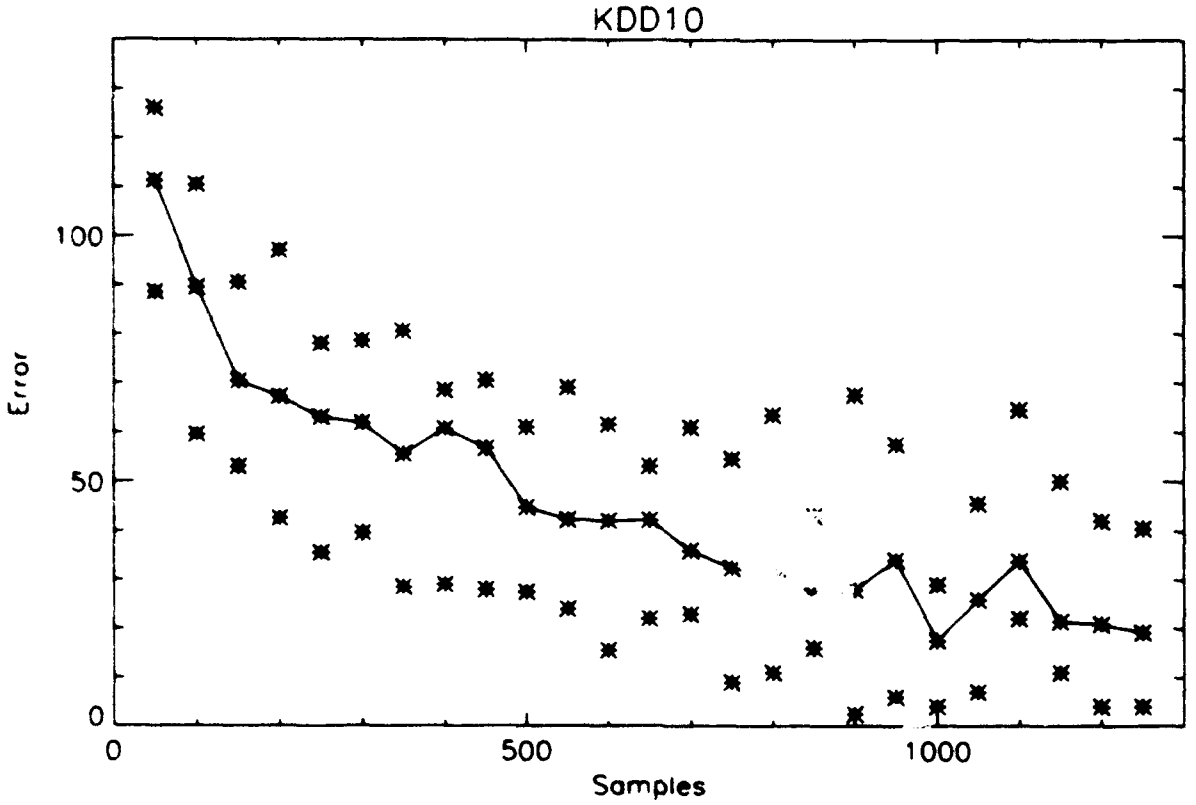
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

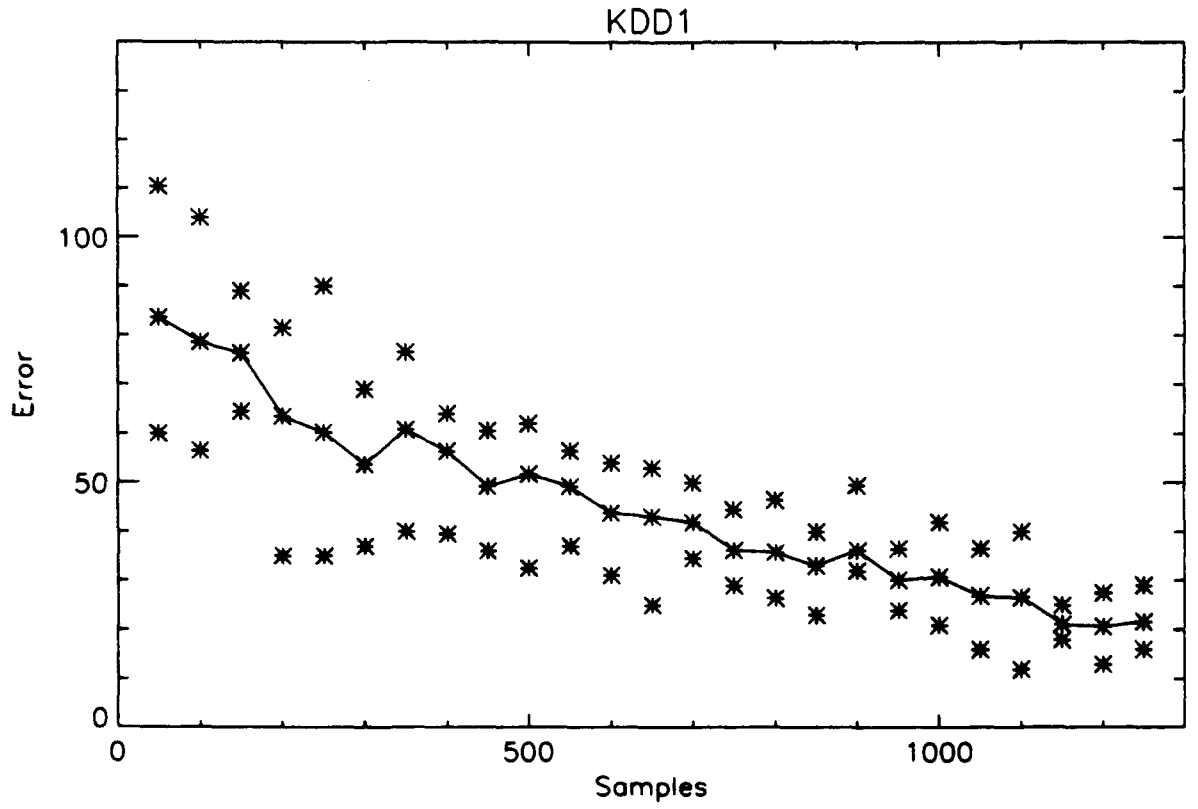
Avg error

dni0e300

sampling with replacement

20% noise

Lose Conflicts and Remove Duplicates (preprocessing)



Max error

Min error

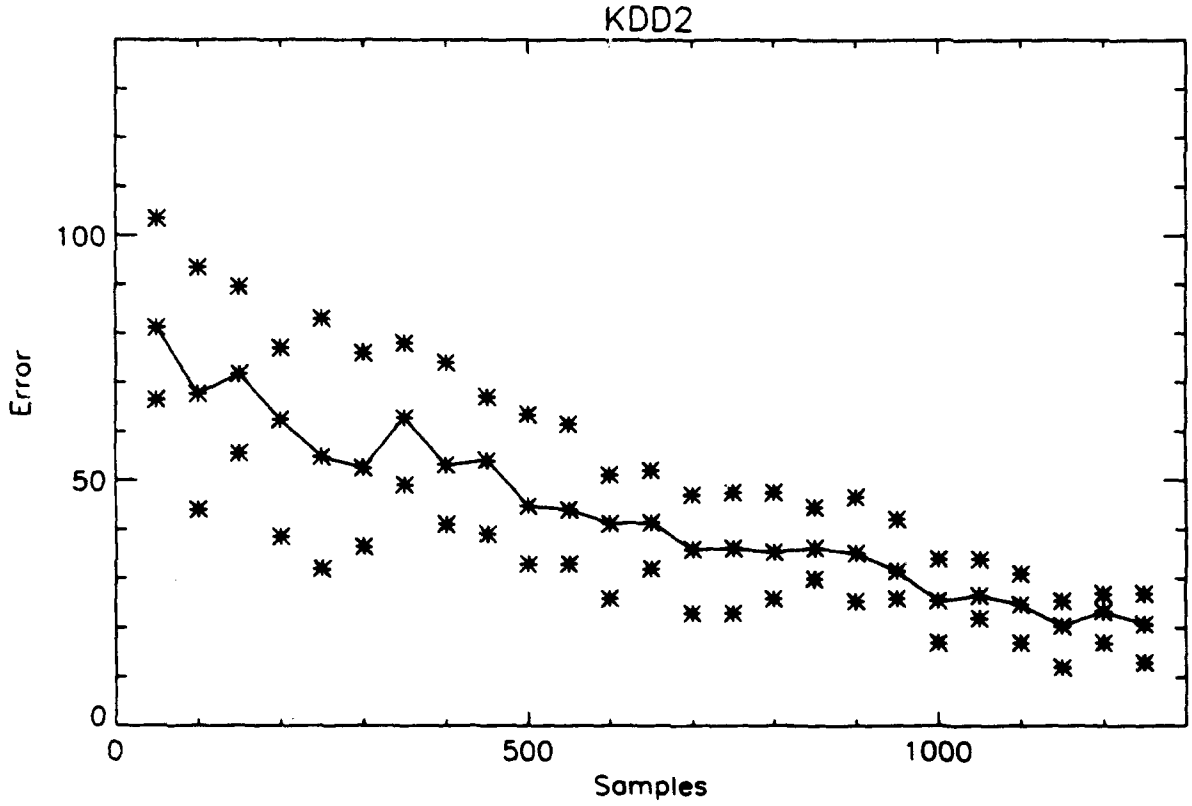
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

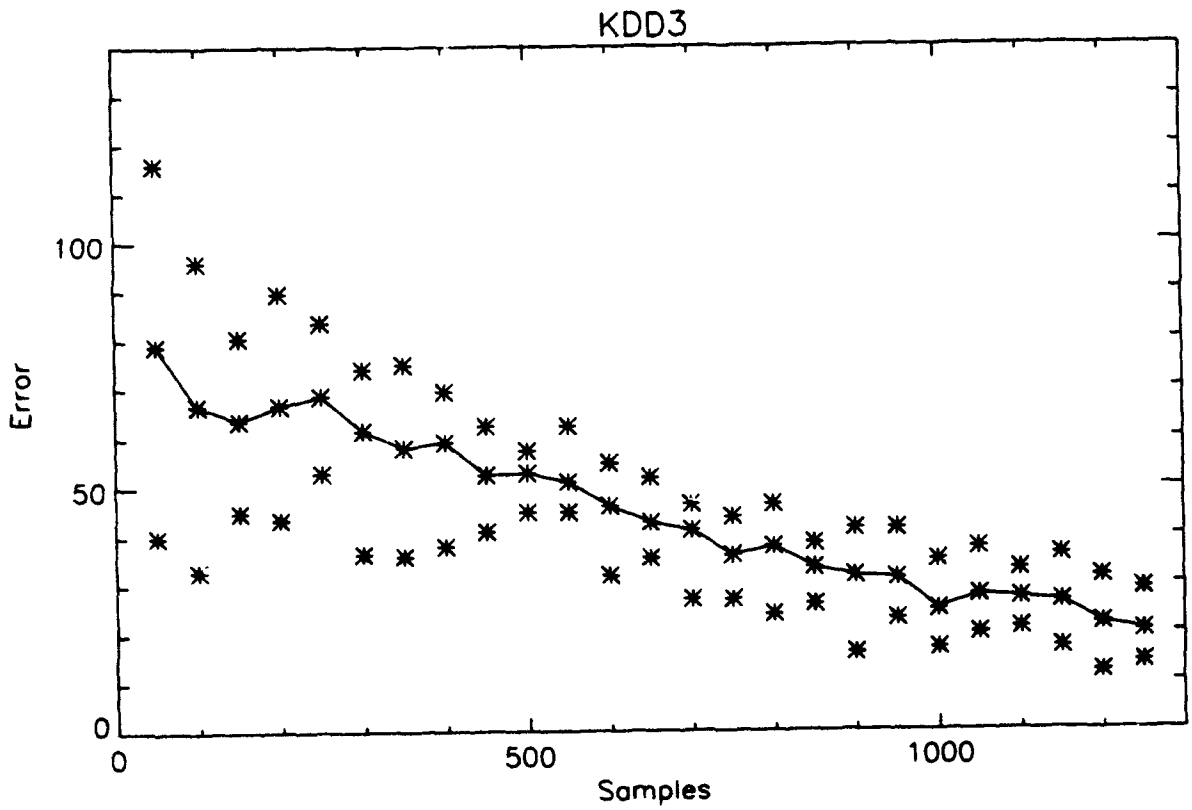
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

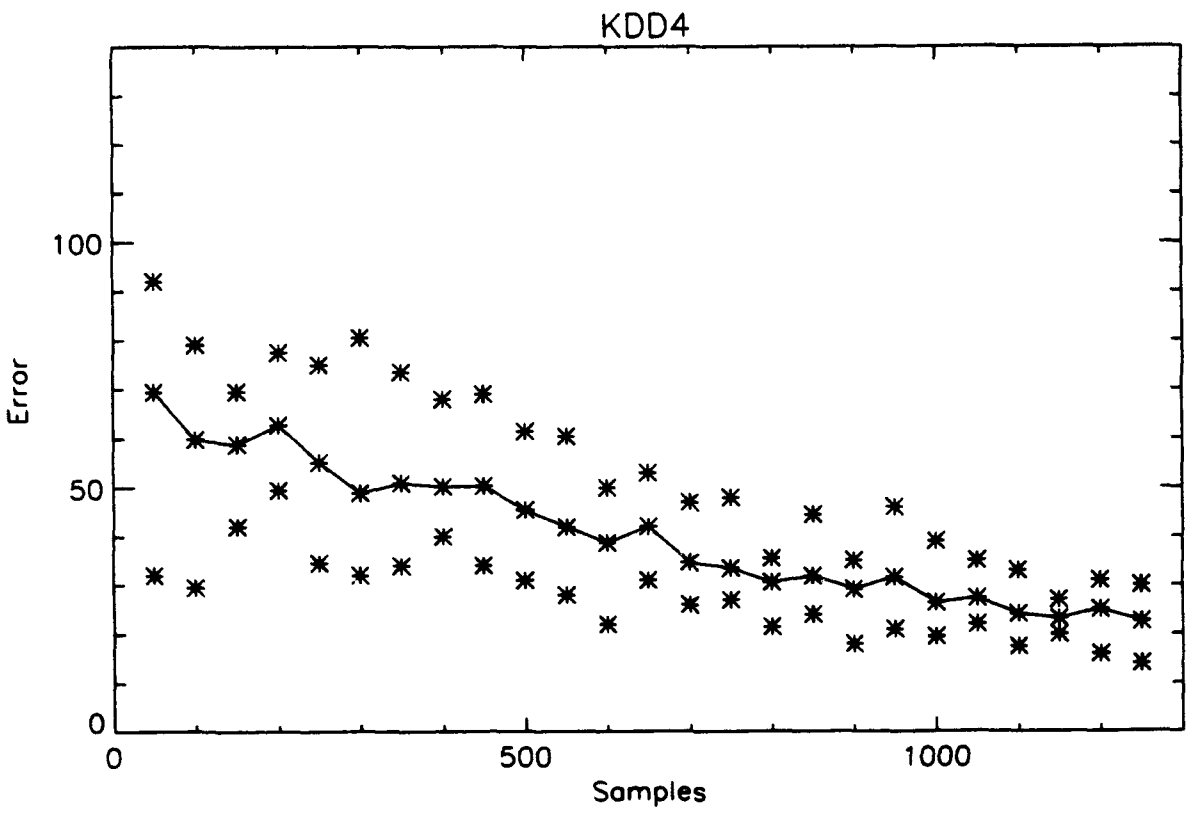
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

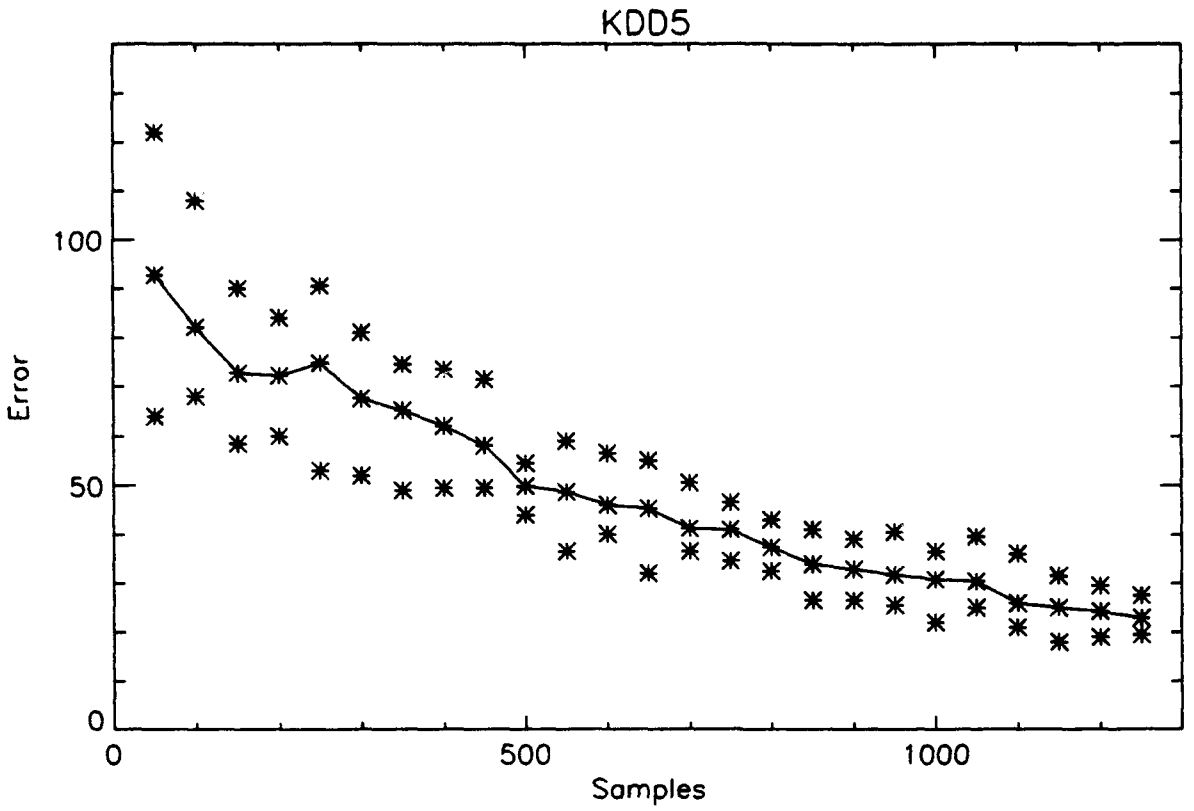
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

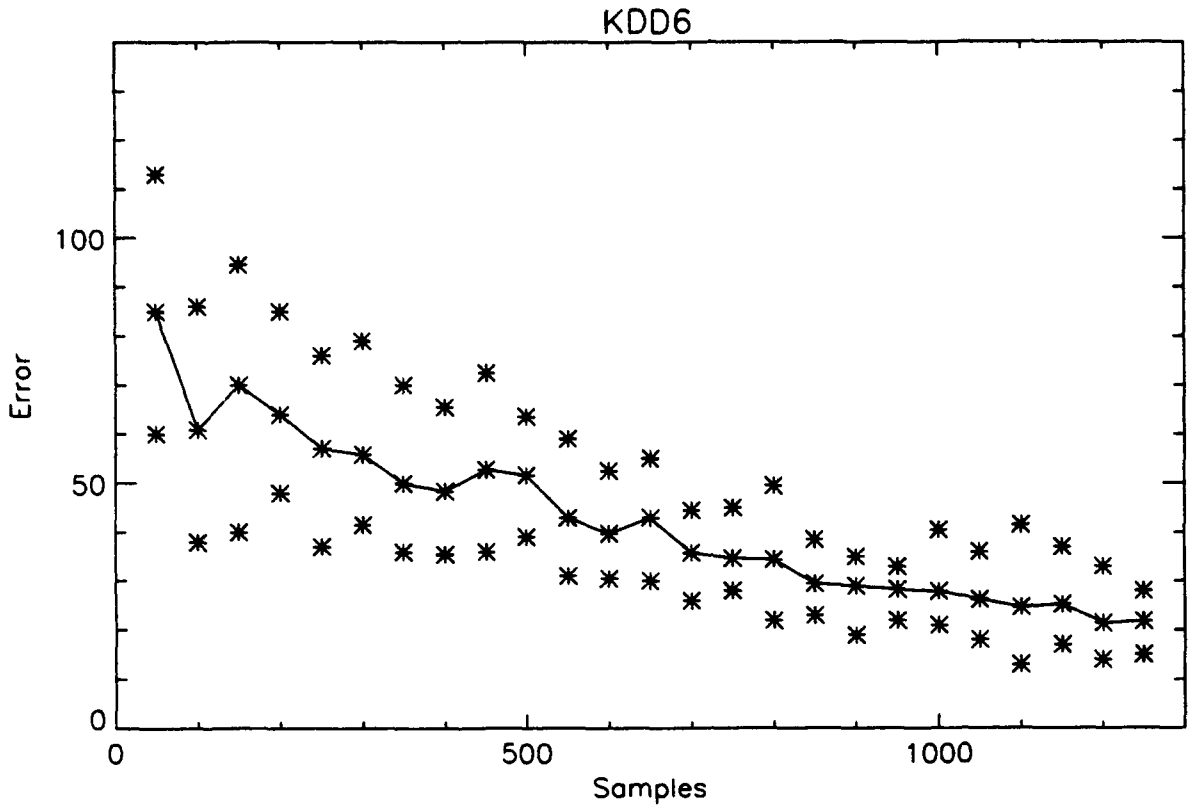
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

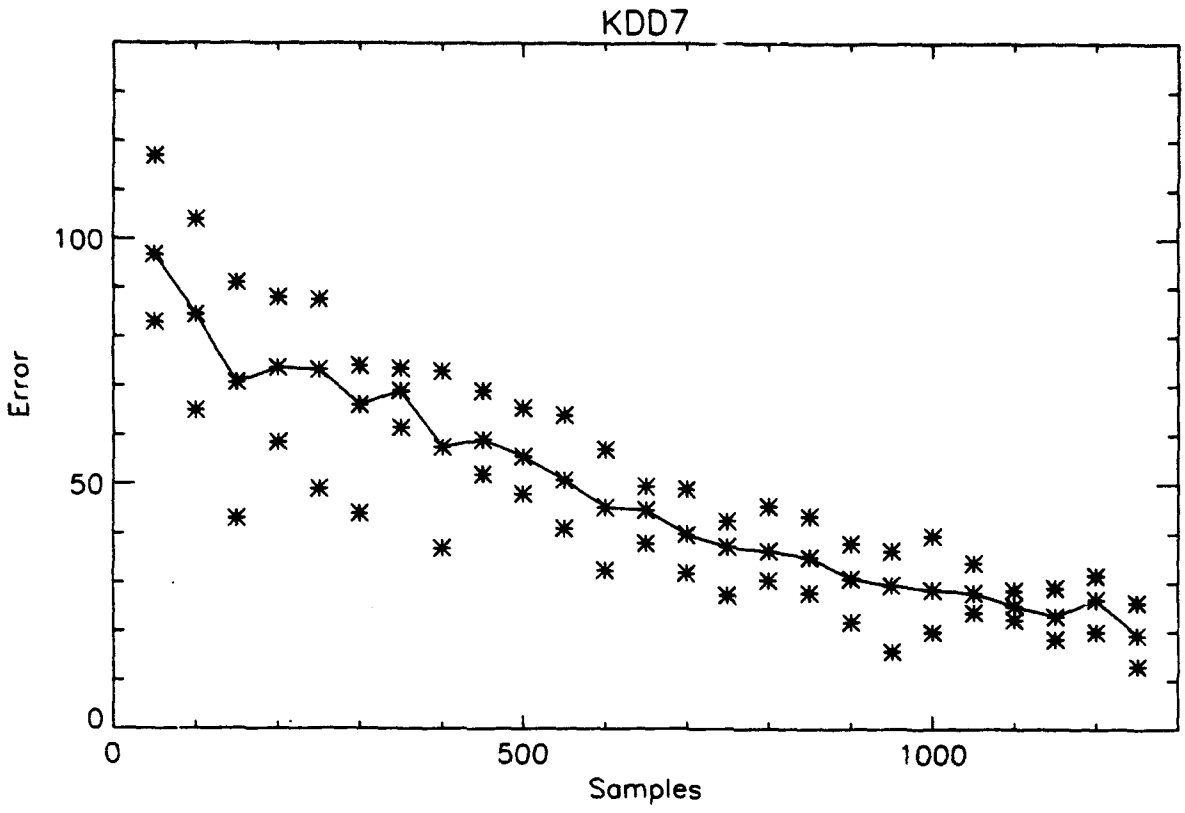
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

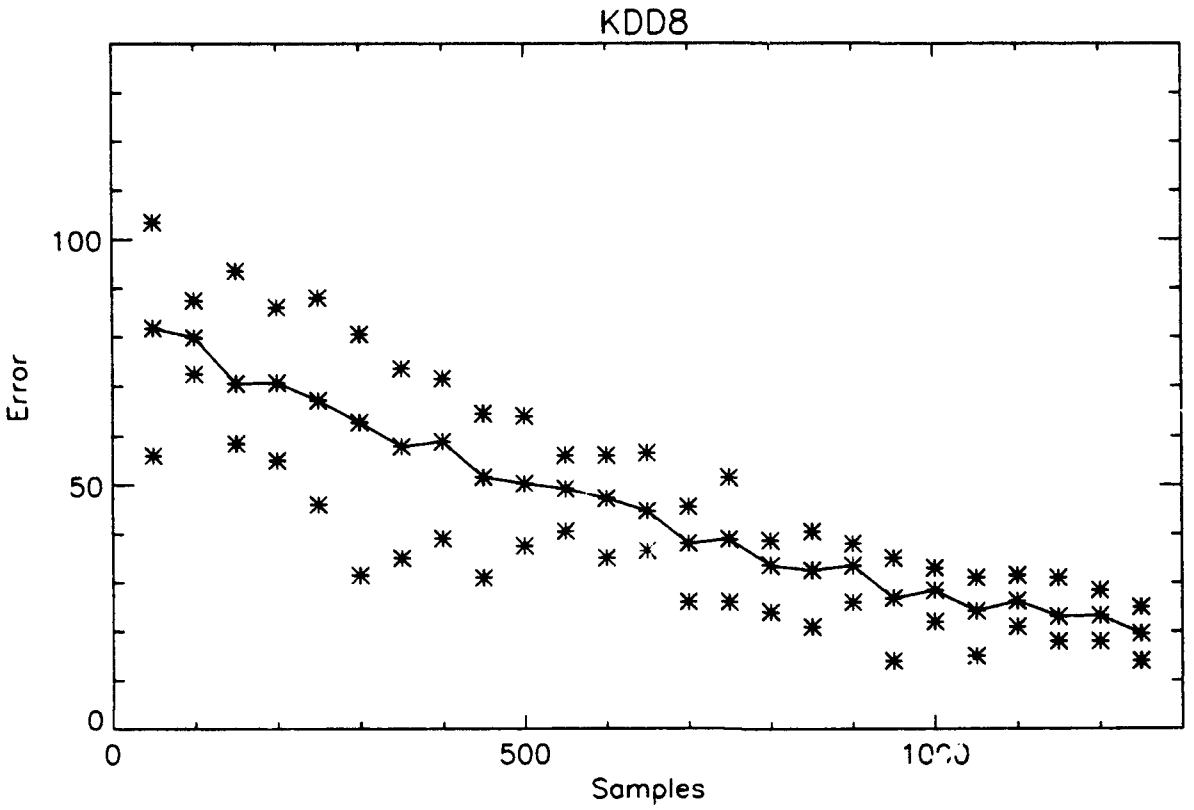
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

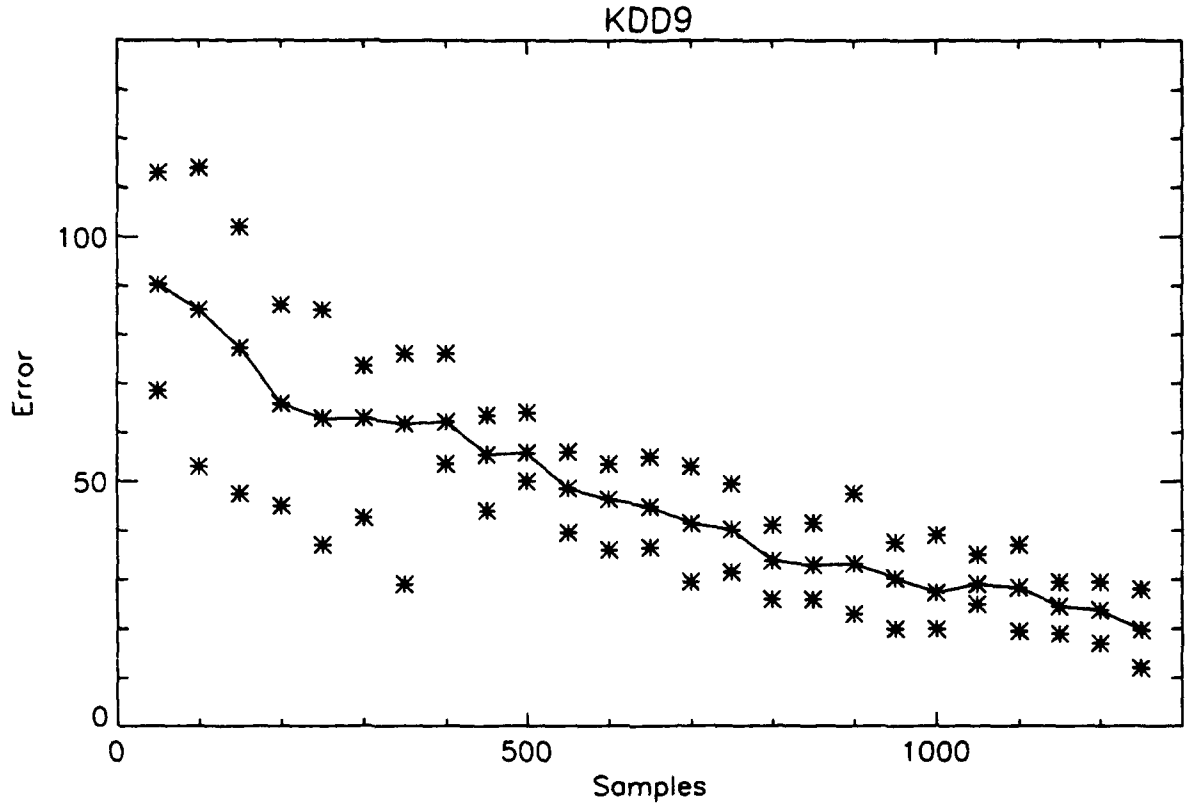
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

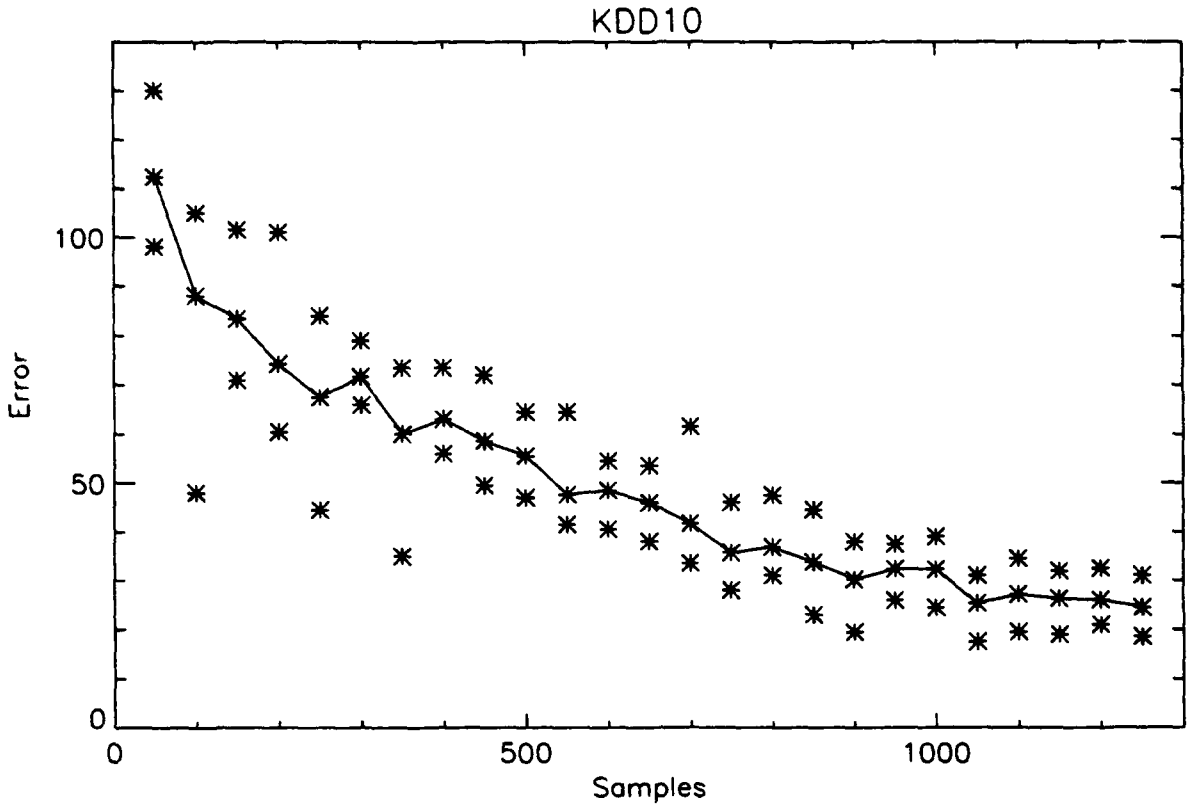
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)



Max error

Min error

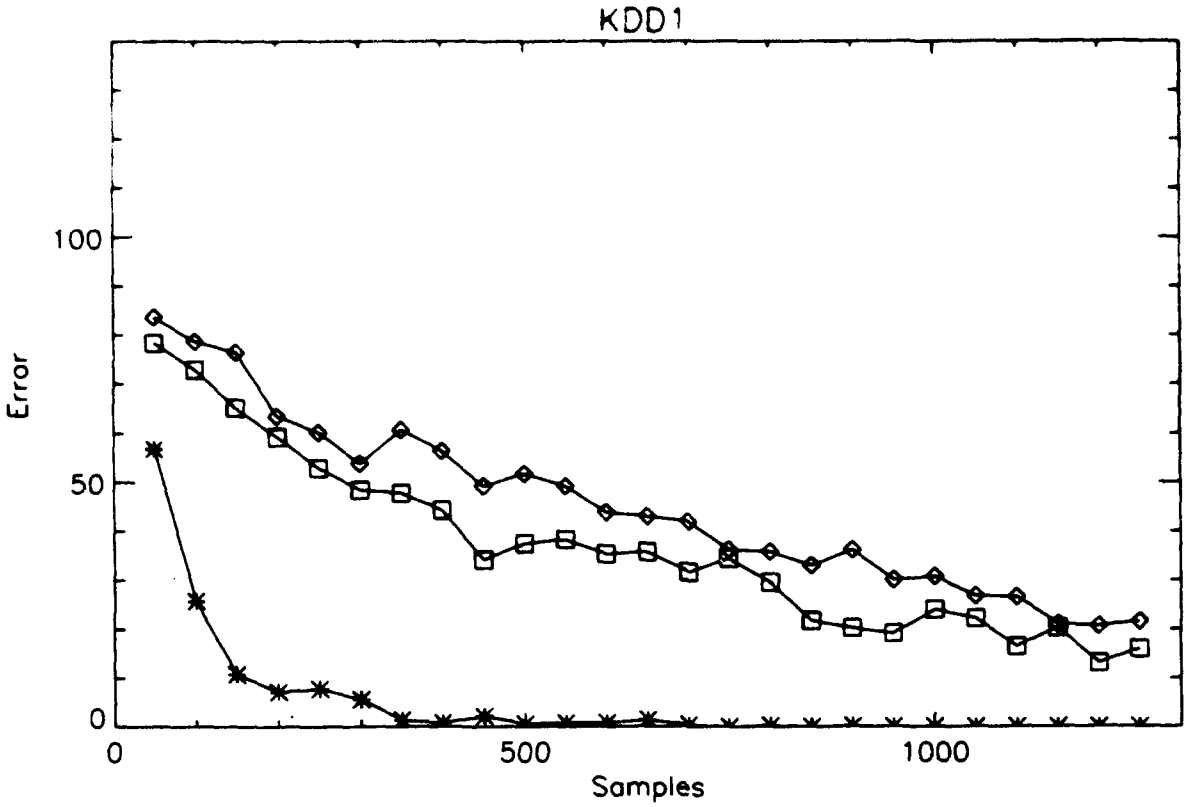
Avg error

dni0e300

sampling with replacement

20% noise

Majority Rules and Remove Duplicates (preprocessing)

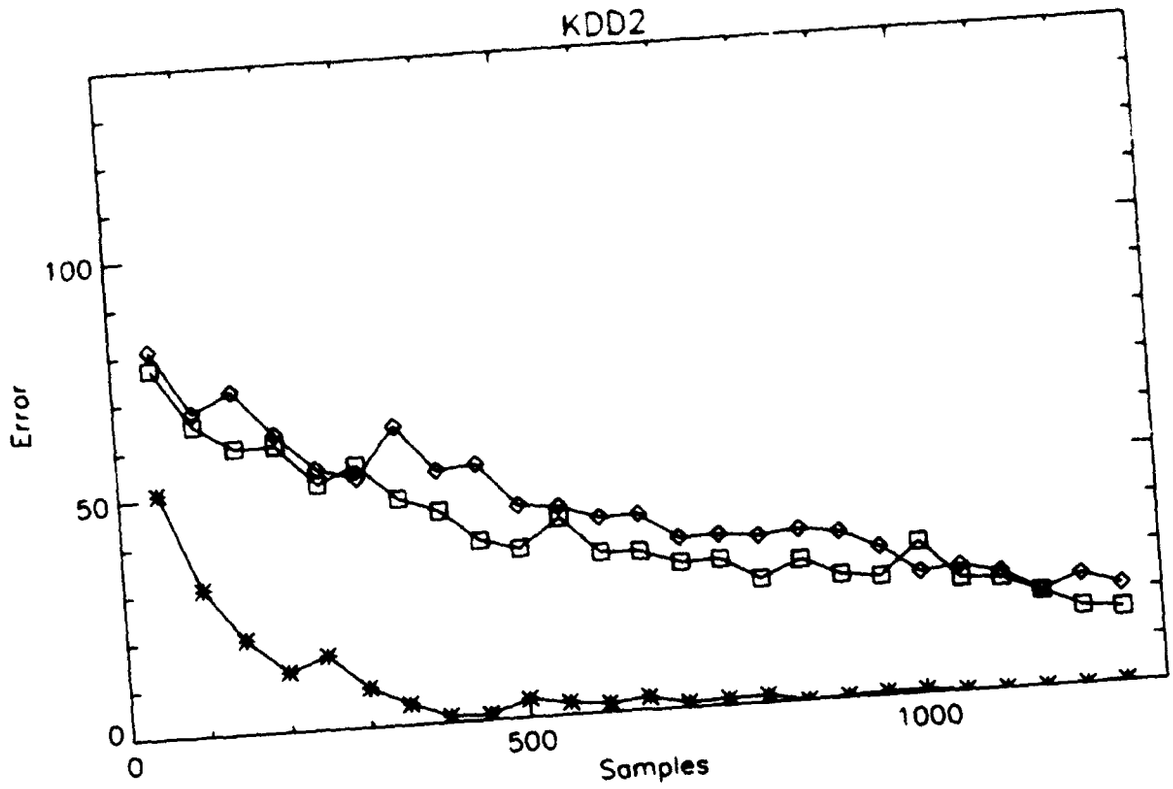


Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 − Lose Conflicts, Remove Duplicates

FLASH dni0e300 − Majority Rules, Remove Duplicates

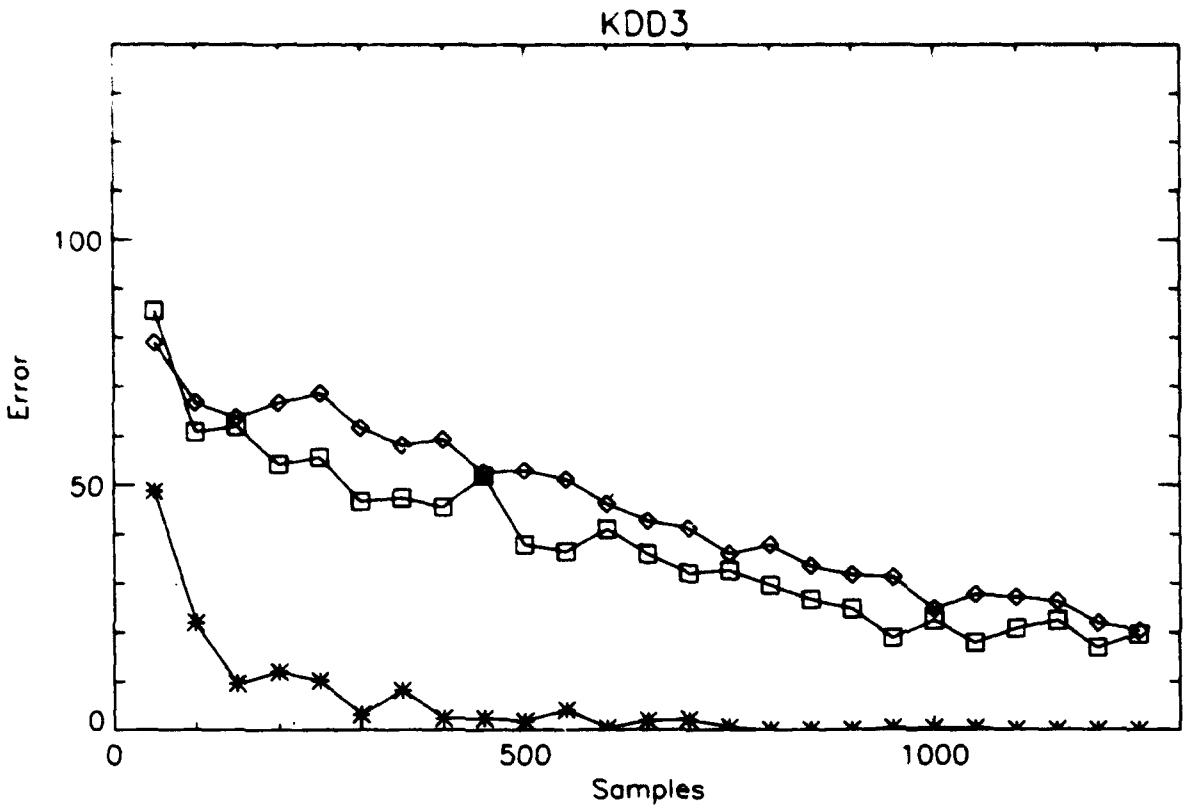


Sample with Replacement, 20% noise

C4.5 (-m 0, -t 10) pruning

FLASH dni0e300 – Lose Conflicts, Remove Duplicates

FLASH dni0e300 – Majority Rules, Remove Duplicates

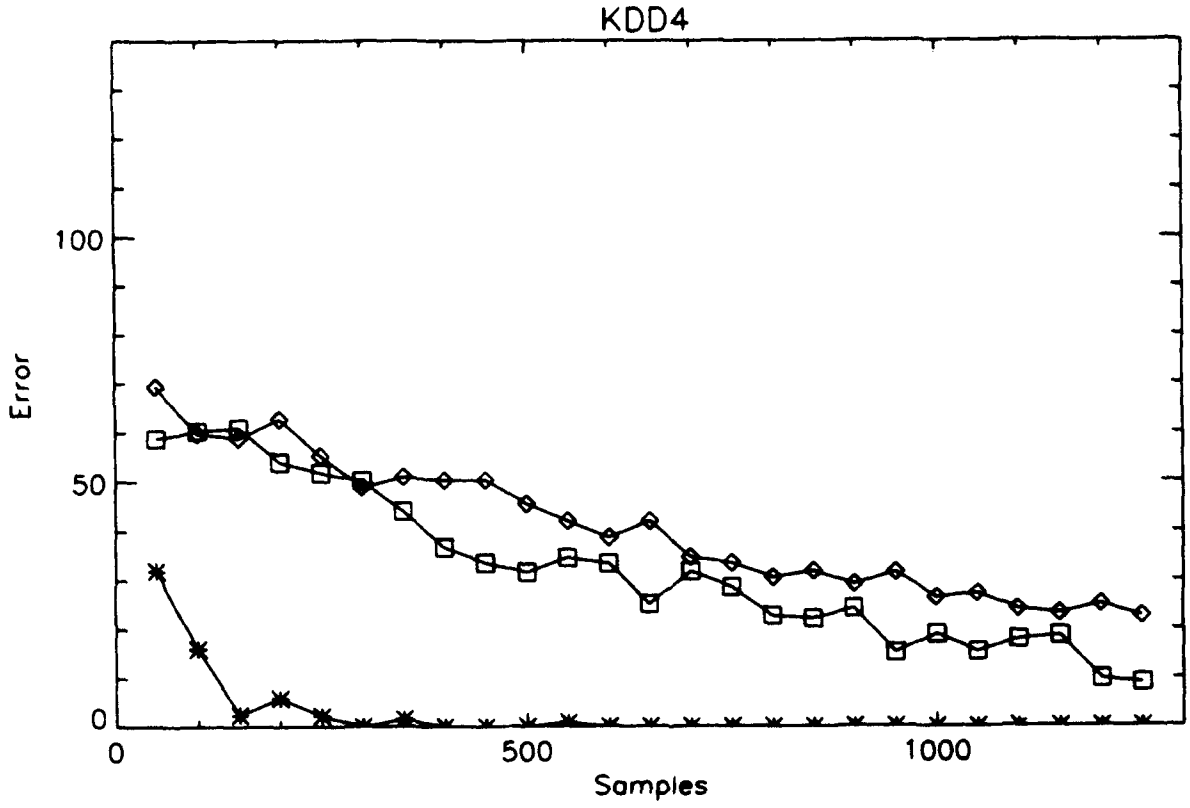


Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 − Lose Conflicts, Remove Duplicates

FLASH dni0e300 − Majority Rules, Remove Duplicates

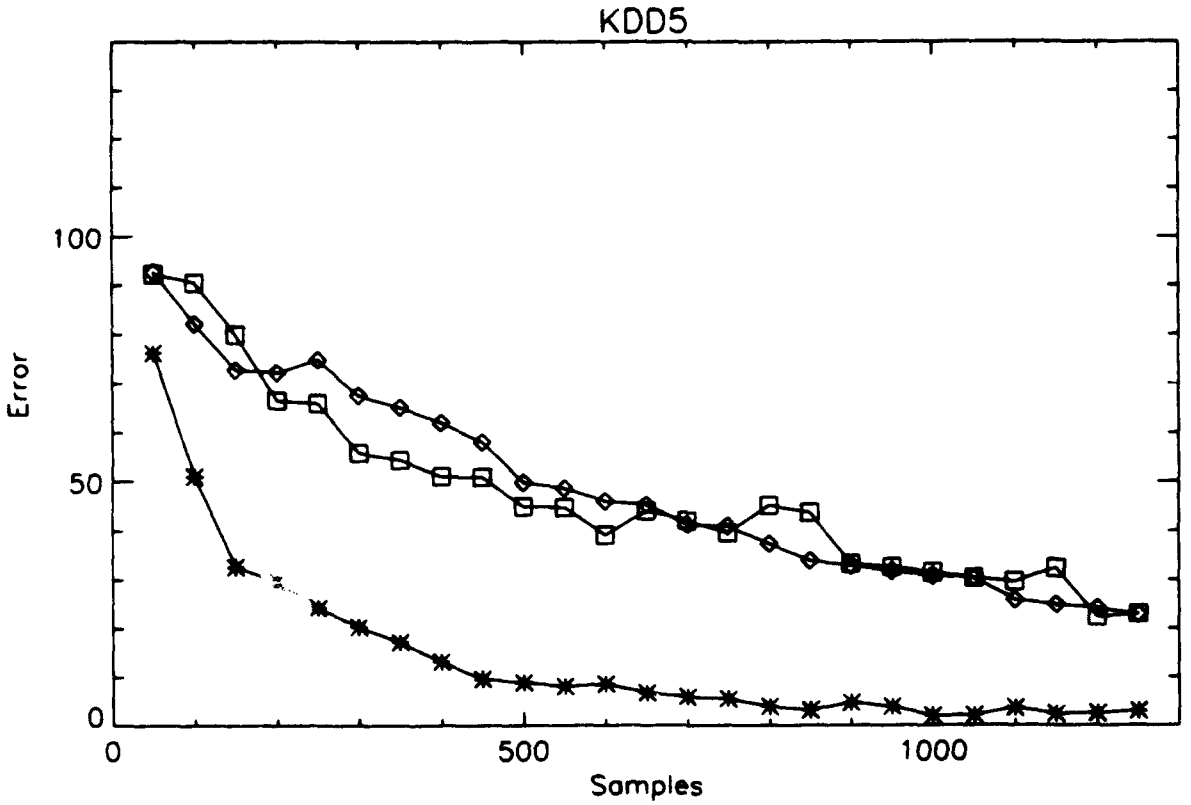


Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 − Lose Conflicts, Remove Duplicates

FLASH dni0e300 − Majority Rules, Remove Duplicates

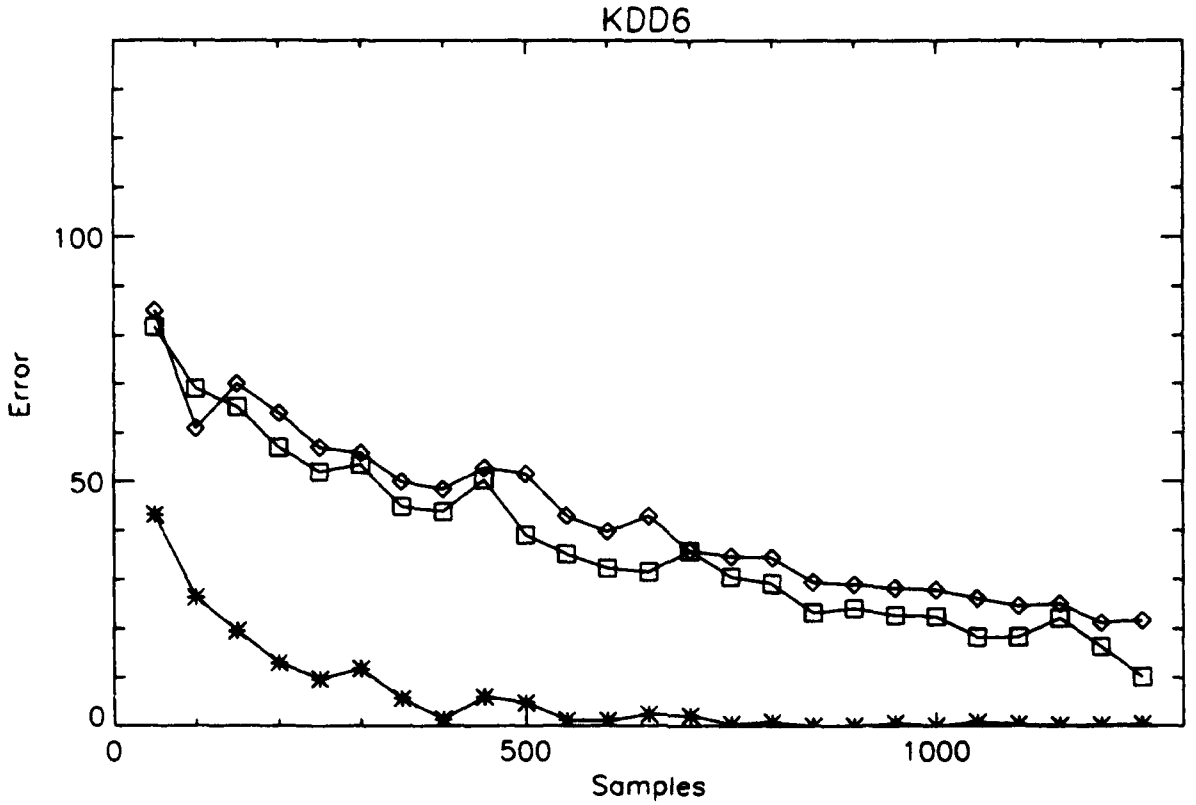


Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 − Lose Conflicts, Remove Duplicates

FLASH dni0e300 − Majority Rules, Remove Duplicates

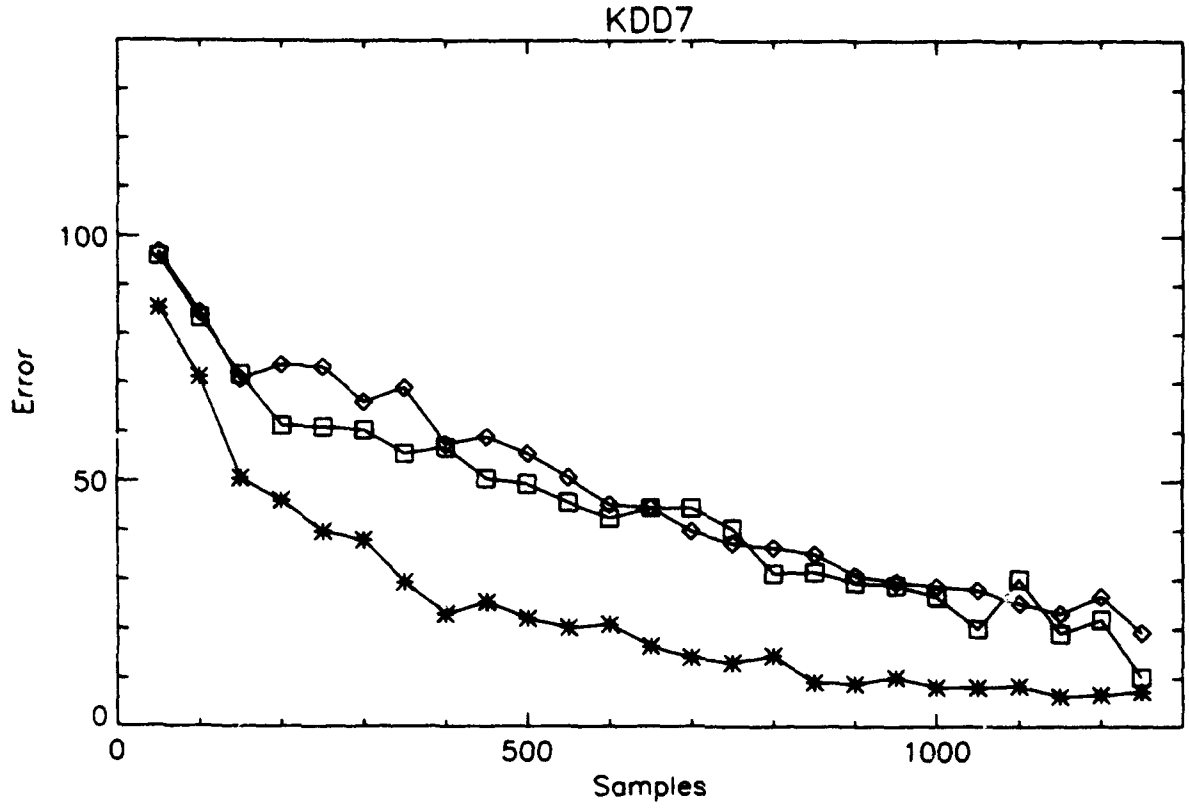


Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 − Lose Conflicts, Remove Duplicates

FLASH dni0e300 − Majority Rules, Remove Duplicates

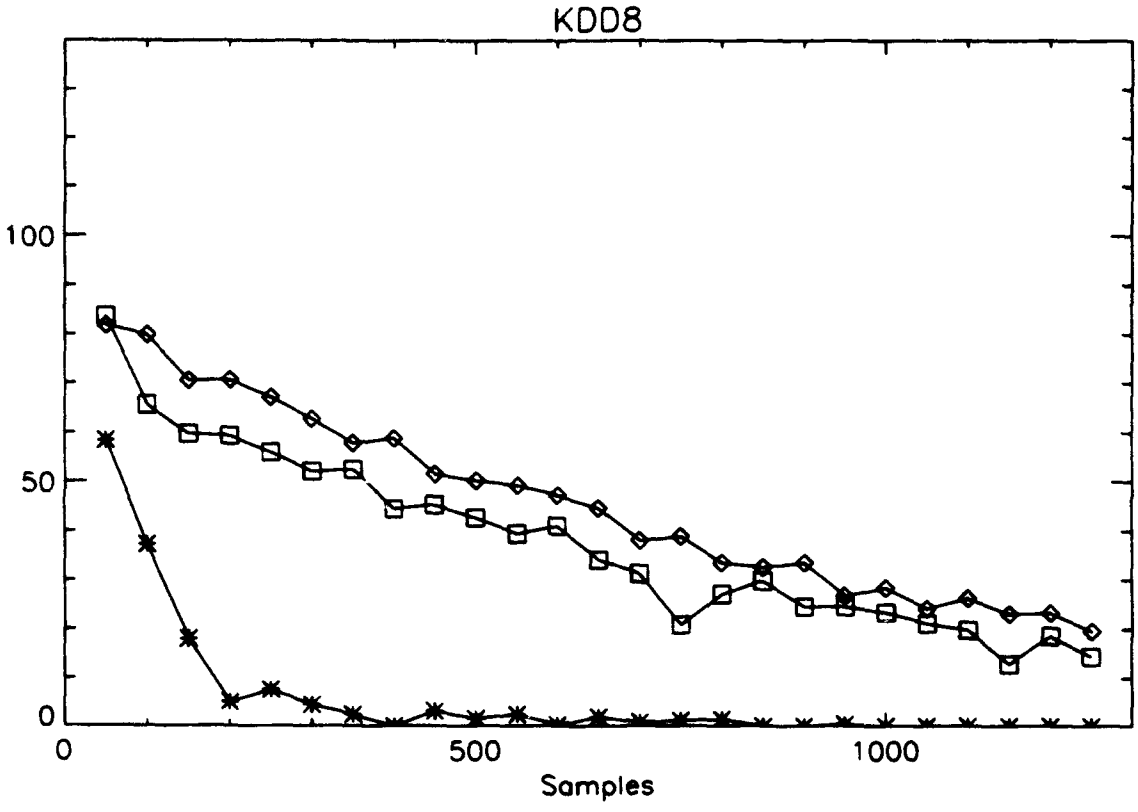


Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 − Lose Conflicts, Remove Duplicates

FLASH dni0e300 − Majority Rules, Remove Duplicates

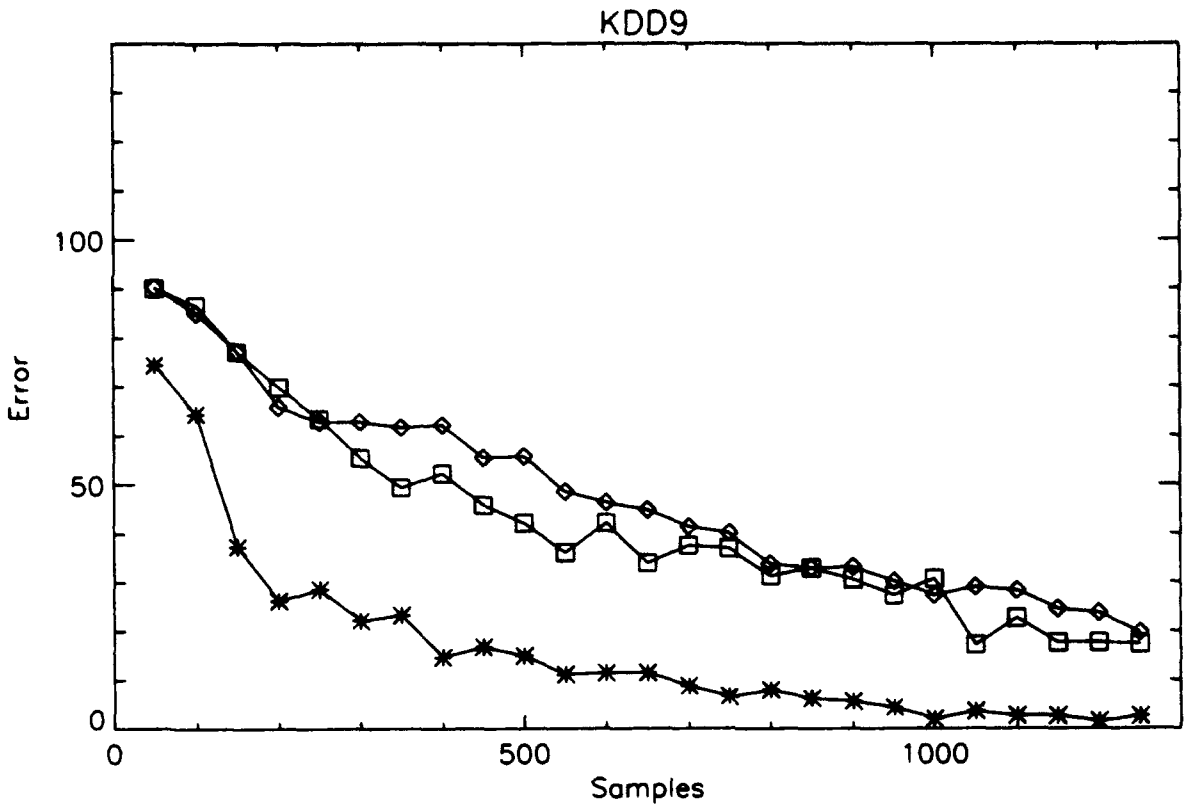


Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 – Lose Conflicts, Remove Duplicates

FLASH dni0e300 – Majority Rules, Remove Duplicates

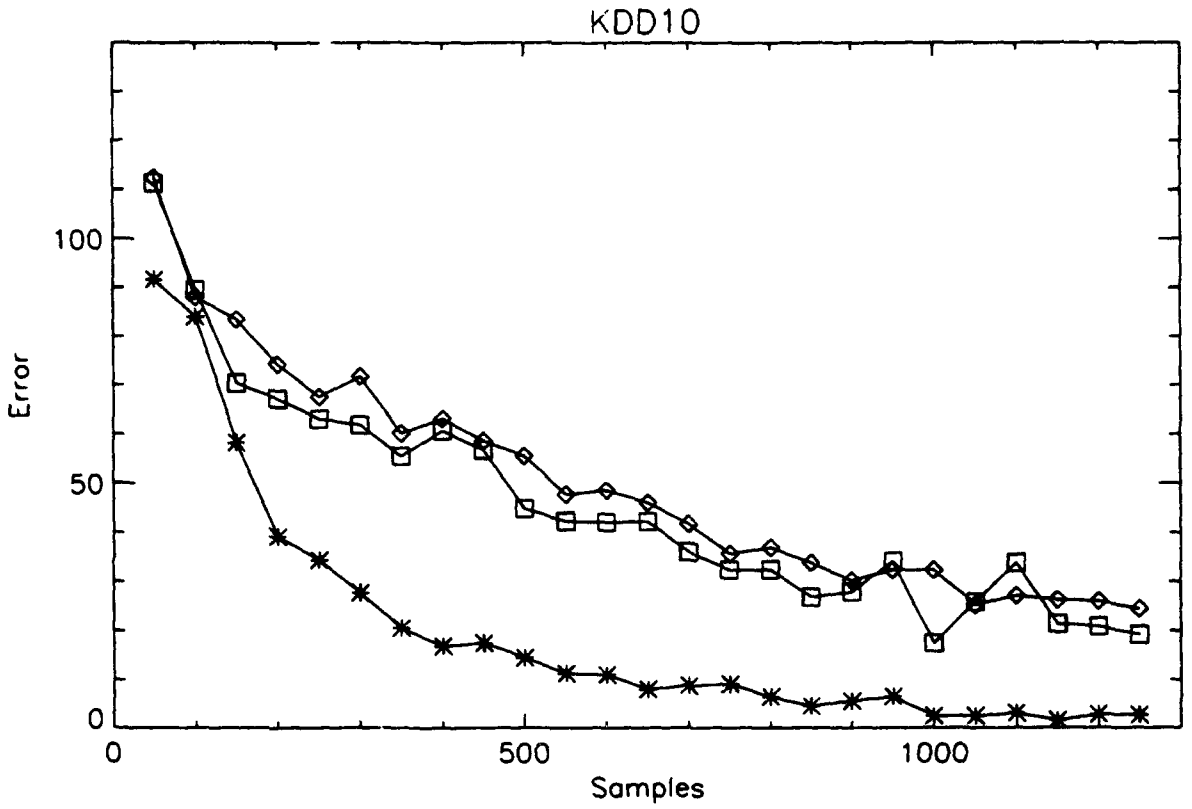


Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 − Lose Conflicts, Remove Duplicates

FLASH dni0e300 − Majority Rules, Remove Duplicates



Sample with Replacement, 20% noise

C4.5 (−m 0, −t 10) pruning

FLASH dni0e300 − Lose Conflicts, Remove Duplicates

FLASH dni0e300 − Majority Rules, Remove Duplicates